MASTERARBEIT

Titel der Masterarbeit

"webAD: Visualizing Algorithms and Data Structures"

Verfasser

Volodimir Begy, BSc

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2015

Studienkennzahl lt. Studienblatt:	A 066 926
Studienrichtung lt. Studienblatt:	Masterstudium Wirtschaftsinformatik
Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta

Eidesstattliche Erklärung
Hiermit erkläre ich eidesstattlich, die vorliegende Arbeit selbstständig und ohne Benutzung anderer, als der angegebenen Quellen und Hilfsmittel verfasst zu haben. Die aus fremden Quellen übernommenen direkten oder indirekten Gedanken, Konzepte und Zitate sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder in gleicher oder ähnlicher Form verfasst, noch einer anderen Prüfungsbehörde vorgelegt.

Wien, September 2015

Unterschrift:

____
Volodimir BEGY

About this thesis

webAD is a web-based visualization platform for Algorithms and Data Structures developed by the research group Workflow Systems and Technology at the University of Vienna. Launched in 2014, its contents are oriented on the materials of the self-titled course *Algorithms and Data Structures*. webAD reuses experiences collected by previous visualization tools of the WST research group, such as VADer or NetLuke and extracts the best from them.
Aimed at support of both students and lecturers, it deals with one of the most important fields of computer science: Algorithms and Data Structures. This topic is the foundation of programming, and the progress of modern world directly depends on it. More efficient data structures and algorithms spare enormous amounts of time in countless daily activities, be it a commercial bank transaction or a biological comparison of DNA genomes. Thus, the significance of fine quality education in this sphere for all computer science students may not be underestimated, any qualified professional should master the knowledge and skills in this domain.
Needless to say, the quality of the product in sense of didactics and usability is the focus of the work, proper material should be explained through appropriate channels, minimizing the cognitive efforts. Consequently, following the trend of cloud technologies and our minimalistic vision, we have developed a robust online platform with intuitive operability, based exceptionally on HTML5 and JavaScript, realizing a fat client with a Model-View-Controller architectural pattern.
This thesis introduces the state of the art of the scientific field, documents and justifies webAD's design and provides developer and user guides.

Acknowledgements
The project's Architect and Lead Developer is Volodimir Begy. My special thanks go to contributors Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta (Scientific Advisor) and Kirilo Begy (Web Designer) for their eminently qualified assistance.

Contents

1 Introduction

1.1 Motivation

"Algorithms $+$ Data Structures $=$ Programs."

— Niklaus Wirth

Scientific methodology is vital for construction of complex high quality software. This set of rules should include guidelines for the way data is organized (data structures) and the way the data is manipulated (algorithms). The choice of implemented algorithms is often dependent on the underlying data structure, on which they have to be applied. The same way around, with given pre-defined algorithms the suitable data structures are chosen accordingly. The second example is rather not trivial, because data structures serve as initial base of programs. The most significant and atomic data structures, such as record or array are interwired with their mathematical definitions [45].

The nature of algorithms is as well fully dependent on mathematic principles. The topic of algorithm complexity is today one of the few most burning issues in the field of computer science. Scientists try to optimize algorithms for a lower complexity. For example the execution time of processing one hundred elements by an algorithm of complexity *O(n)* or $O(2^n)$ is enormously different. The progress of modern world directly depends on algorithms and their execution complexities, because today many actual topics, like comparison of DNA genomes take too long because of complexity limitations.

At the University of Vienna every undergraduate informatics student has to take the course *Algorithms and Data Structures*. Understanding its topics is highly important for the qualification, since algorithms and data structures are the backbone of programming and thus our daily life. Lecturers mainly use predefined examples from slides, which have the flaw of potentially false interpretation due to the lack of the possibility of trying out any desired combination of numbers. Last, but not least, computer science students are rather accustomed to e-learning. Considering this, and the trend of the cloud technologies, which destines applications to run through the browser, there is a natural demand for an online visualizer.

In Winter 2013 I have started to contribute to NetLuke [5], which was the most actual project handling the issue of the visualization of algorithms and data structures by WST research group. NetLuke is implemented with a complex server side architecture. After discussions with Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta we have decided to launch a new project, which would go back to the roots and build exceptionally on JavaScript and HTML5, forming a fat client for most simple and minimalistic use by students.

1.2 The Field of E-Learning

E-learning is the art of conveying the teachings through a digital gadget [37]. It represents an appropriate channel to explain algorithms and data structures because of several factors. First of all, it enables self-study, which is vital for a university course. Furthermore, the chosen web technologies make it is easily accessible for students and teachers anywhere, be it at home or in a classroom. At last, the most important feature is the effectiveness of the e-learning.
The first empirical survey in the year 1947 was conducted by the U.S. Army and examined three channels of instructing, all of them delivering the same information: digital learning by showing a movie, typical class with a teacher and a self-study with a paper script analog to the one shown in the film. The outcomes of this and hundreds of analog experiments have shown, that in average the classical and digital teaching results in same achievement effect. In rare cases e-learning provides better results than traditional learning and vice versa. The described data forms a normal distribution, depicted in figure 1 [37].

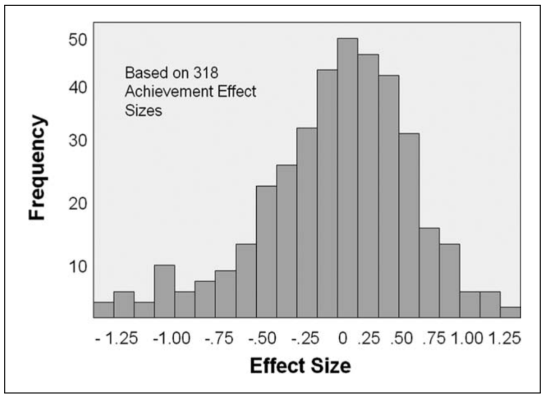


Figure 1: Effectiveness of E-Learning Versus Traditional Learning [37]

This proves, that generally e-learning has no disadvantages in sense of delivering information, while in specific cases it is much more effective. Visualization of algorithms and data structures is one of such rare cases, because it is based on animation and interaction.
It is important to follow proven guidelines while designing an e-learning tool.

The book [37] introduces tips for designing a digital learning platform based on visual teaching only. The following list sums up the essence of the hints, which were significant for webAD, linking the bullet points to the platform:

- Depict only relevant content using animations and primitive visuals for cognitive offload. This is why webAD is kept minimalstic.
- Include pre-defined examples to avoid false interpretations. When a certain topic is opened, a hardcoded example appears. Users can also generate random instances.
- Provide ability to manipulate the animation. The tape-recorder of our platform is highly integrated into the platform's workflow. Several modules contain tunable settings.
- Proper segmentation of contents. The tool sorts all modules into didactic topics.
- Provide additional information and help. webAD includes *Info* and *About* sections.

Few minor hints were ignored, because they did not comply with a visualization tool for a university course. For example, it is advised to use first and second person while writing texts, to reach the personalization effect [37]. This does not necessarily conform to a tool used in a scientific field.

1.3 Previous Work

webAD, standing for web-based visualization of Algorithms and Data Structures was launched in 2014 as my bachelor thesis.
The first document described the problem statement, main design decisions based on functional and non-functional requirements, tools and included the brief prototype proving the concept.
Functional requirements have identified, what modules need to be implemented, sorted into chapters, and what operations they should visualize. The choice of chapters was made precisely, since they are still maintained. The structures of the modules and their operations have been modified after iterations of re-design.
The non-functional requirements reflected the vision of the project and included following [36]:

- Extensibility of the framework.
- Accessibility, compliance and security.
- Fault tolerance and testability.
- MVC architectural pattern.

Realization of webAD was oriented on these requirements.

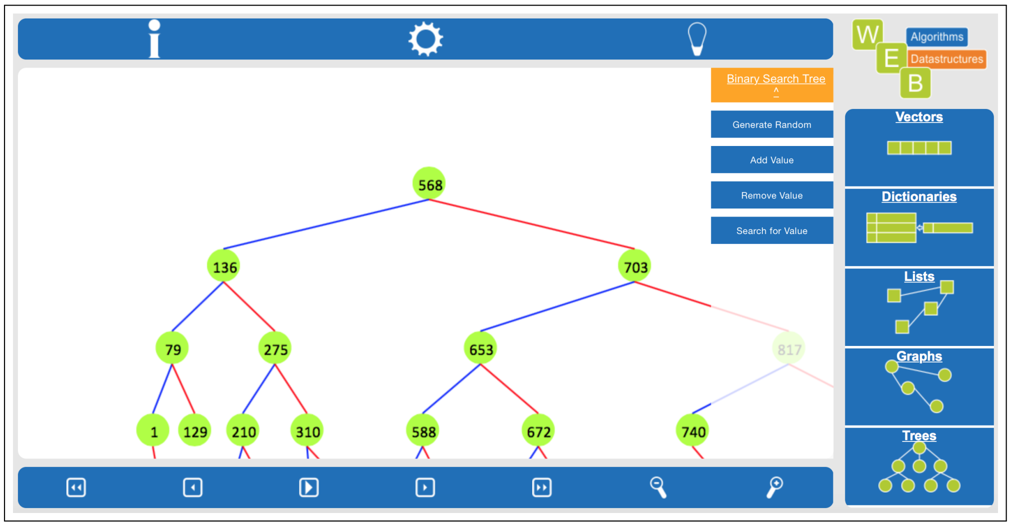


Figure 2: Prototype from 2014

Proof of concept implemented first versions of Bubble Sort, Linear Probing and Binary Search Tree. Development iterations have improved all of them significantly in following ways:

- Bubble Sort proceeded to next pair of elements after the swap immediately. Now the visualization is more clear, if the elements get switched an extra step of animation represents it.
- Linear Probing had issues with timers (the animation time was not consistent) and provided only the static version. The module visualized only the creation of the table and the insertion of a value. In actual version the animation is corrected, the extendible version of the table has been added and user may also visualize removal and search for values.

- Binary Search Tree had issues with the view, starting with the depth 4 the nodes began to intersect. The new view algorithm extends the tree without limits and intersections of the vertices.

The design of the website was renovated using expert visualization rules, providing better cognitive offload. Large areas use smooth and pastel colors, while smaller areas are marked with more bright and saturated colors [40]. Furthermore the homepage was constructed.
With large expansion of the components the file hierarchy was re-organized in a suitable manner.
The design decisions have proven to be chosen reasonably, because the implementation of the system's core went smoothly as planed and only minor things have been adjusted.

1.4 Goals

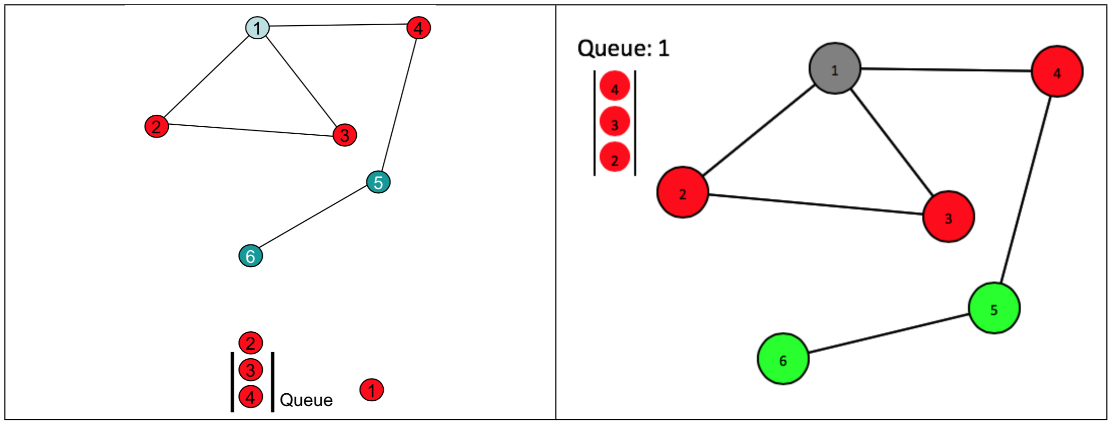


Figure 3: Comparison of Lecture Slides and webAD's Visualization [39]

From the ideological point of view the two main goals of the project are:

- Coupling with and reflection of didactic materials of the *Algorithms and Data Structures* course.

- Minimalism, clear structure and simple use for the cognitive offload (look & feel). Though it can be argumented, that additional components within the system, such as surfacing popups with quizzes or pseudocode view are helpful for learning, we believe, that they rather discourage the user when overflown, causing cognitive tension: the more components of the interface are integrated into the workflow, the less overview remains. This is why in webAD for example a general pseudocode is stored together with theoretical information in the *Info* window.

Both tasks are accomplished, the visualization reflects lecture slides (figure 3) and the user interface is intuitive and without any redundant elements (see section *6.2 User Tutorial*).
Most goals form the information system perspective have been defined in the bachelor thesis [36]. webAD is aimed at:

- Extensibility of the framework by other developers. Currently some students are interested in implementing new modules for webAD. Time will show, how well this task is achieved.
- No installation or configuration effort. The browser alone handles the execution of JavaScript.
- MVC architectural pattern. The platform's MVC realization has proven to be well implemented, because the tasks of the Model, View and Controller operate without overlappings and conflicts.
- Fat client, which means the application is working without internet once loaded. This is the case, because webAD has a client side architecture.
- Multi device support. The program is currently running on any device within any browser.
- Independence from external tools and libraries. Throughout iterations of re-design webAD has got rid of several previously used libraries (TaffyDB [10], Backbone.js [9]), because we value code independence, try to avoid issues, that may arise in case a certain library is not maintained anymore, and because we believe, that a system consisting of native components provides a clear overview of the architecture, and thus supports the minimalistic ideology.
- Innovative flexible tape-recorder, which allows user to perform any operation on any chosen state. Even though many visualization tools contain a tape-recorder, they are very limited (selected cases are described in section *2.5 Comparison*). webAD's tape-recorder is a major

part of the platform. It supports the user in any possible way: the animation may be played automatically or paused and switched with separate mouse clicks. The states can be switched not only back and forth in single steps, but also to the very first and last states.

The next sections of the thesis show, that all these goals have been fulfilled. The goals for future contributors are provision of new modules, maintenance of the application and further realization of speech recognition, if it will be requested and desired by users.

1.5 About webAD

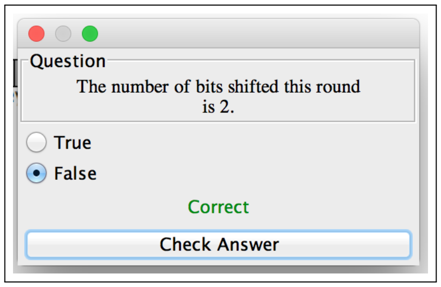
webAD provides 14 modules, fully operable on desktop and mobile devices. The project homepage is located under following url:
`http://gruppe.wst.univie.ac.at/workgroups/webAD/`
The application can be downloaded in form of an archive from the homepage. The GitHub repository for developers can be found at:
`https://github.com/VolodimirBegy/webAD`

2 State of the Art

webAD aims specifically at the needs of computer science students of the University of Vienna. As stated above, our mantra is minimalism, no installation or configuration effort, multi device support, clear structure of didactical content and simple extensibility for developers. Needless to say, there are numerous other visualization tools, which vary in their content and functionality. AlgoViz.org [1] is a scientific portal uniting such projects. In this section selected examples from the portal's Hall of Fame (JHAVÉ [2], Algorithms in Action [3]) and previous projects supervised by Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta (VADer [43], NetLuke) are analyzed in context of didactics and usability.
These tools are selected for the analysis and comparison because of numerous reasons. First of all, JHAVÉ and Algorithms in Action were awarded by AlgoViz for their implementations. Thus, it is interesting to examine one of the best programs in the portal's ratings. VADer and NetLuke are chosen, because they reflect the paving of the path from first to last attempts to implement such a tool by the research group Workflow Systems and Technology. Another reason for the choice of NetLuke, is that I have contributed to it before launching webAD and have gathered an insight into it. At last, all of the tools together with webAD form a wide spectrum of completely different architectures and technologies, which makes it interesting to compare heterogenous products.

2.1 JHAVÉ

Run mainly by students of University of Wisconsin at Oshkosh, JHAVÉ is a Java program. It has received the 2011 AlgoViz Award for the visualization of Sutherland-Hodgman Clipping Algorithm [1].
The application can be launched with WebStart through a jnlp file or locally from jar archive. Both methods require preliminary installation and basic knowledge of Java and are not aimed at mobile devices. When launching the application through the jnlp file, it is still downloaded and verified, the


Question

The number of bits shifted this round is 2.

True [✓ False]

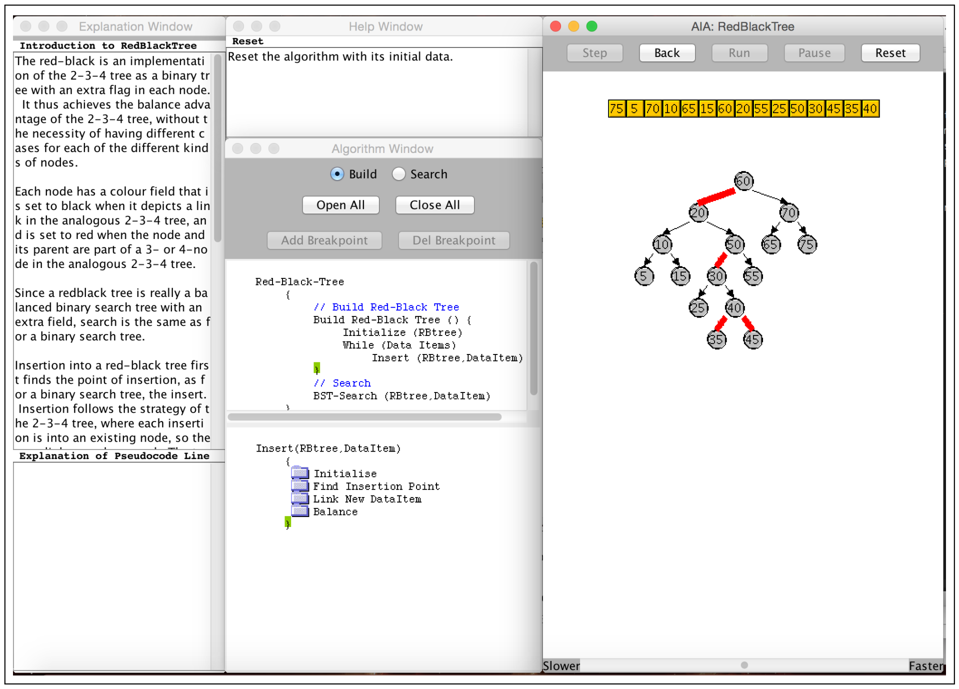
Correct

Figure 4: JHAVÉ Quiz [2]

Java security settings need to be adjusted
to grant it the access. In both cases user needs to connect to the server to retrieve the materials for the visualization, which also takes some effort. There are 32 topics implemented, which is outstanding. Unfortunately they are not sorted by chapters and are just listed one after another. Some of them are Red-Black Tree, Shell Sort, Prim Minimal Spanning Tree, etc. The application aims at teaching and thus provides pseudocode, info, tape-recorder and even interactive popups with questions.

2.2 Algorithms in Action

Algorithms in Action is a project of University of Melbourne. 2011 it was awarded by AlgoViz for its visualization of the 2-3-4 Tree [1]. The homepage supports English and Italian languages.



Introduction to RedBlackTree

The red-black is an implementati
on of the 2-3-4 tree as a binary tr
ee with an extra flag in each node.
It thus achieves the balance adva
ntage of the 2-3-4 tree, without t
he necessity of having different c
ases for each of the different kind
s of nodes.

Each node has a colour field that i
s set to black when it depicts a lin
k in the analogous 2-3-4 tree, an
d is set to red when the node and
its parent are part of a 3- or 4-no
de in the analogous 2-3-4 tree.

Since a redblack tree is really a ba
lanced binary search tree with an
extra field, search is the same as f
or a binary search tree.

Insertion into a red-black tree firs
t finds the point of insertion, as f
or a binary search tree, the insert.
Insertion follows the strategy of t
he 2-3-4 tree, where each inserti
on is into an existing node, so the

Explanation of Pseudocode Line

Reset

Reset the algorithm with its initial data.

```
Red-Black-Tree
    {
        // Build Red-Black Tree
        Build Red-Black Tree () {
            Initialize (RBtree)
            While (Data Items)
                Insert (RBtree,DataItem)
        }
        // Search
        BST-Search (RBtree,DataItem)
    }
```

```
Insert(RBtree,DataItem)
    {
     Initialise
     Find Insertion Point
     Link New DataItem
     Balance
    }
```

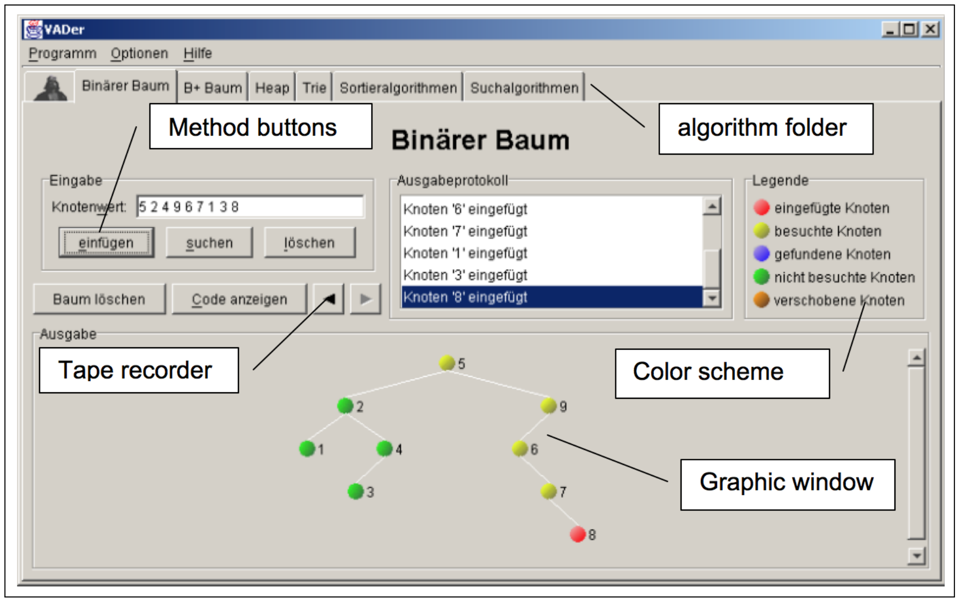
Figure 5: Operating Windows of Algorithms in Action [3]

The application is launched in browser, it is not downloadable, requiring JavaScript and Java (for Java the security settings are required to be configured). When executed, the application is split into four windows: al-

gorithm, pseudocode, explanation and help windows. The homepage is not integrated with them and needs to be opened through a separate fifth window. This is not optimal for mobile devices.
Otherwise this project is well adapted for teaching. There are 33 topics implemented, which are hierarchically sorted into corresponding chapters, delivering a better overview. The interface provides a tape-recorder, pseudocode, info and a debugger with breakpoints.

2.3 VADer

Published in 1999 VADer was the first project by the research group Workflow Systems and Technology in the field of algorithm and data structure visualization. It was a web-based Java program enthusiastically welcomed by the students of University of Vienna.



```
Knoten '6' eingefügt
Knoten '7' eingefügt
Knoten '1' eingefügt
Knoten '3' eingefügt
Knoten '8' eingefügt
```

Figure 6: VADer Interface [43]

The interface (Figure 6) contains handy features for the learning, such as tape-recorder, legend and output protocol. Almost no cognitive effort arises while using the tool, the GUI sections are well placed among one window. It would be more optimal to assign the implemented modules into homogenous chapters.

Since VADer was a young project, there were 10 algorithms/data structures realized, but with gained acclaim among the users the extensibility plans looked promising. The critical issue was the release of new version of Java, which was drastically differing and made the maintenance of the program too complex. Due to this the project was unfortunately stopped.

2.4 NetLuke

NetLuke is the last project before webAD handling the issue of visualization. Its speciality is the heavy use of web technologies with a server-side Java implementation. This design decision was made in order to be able to support user accounts and persistent sessions. Though these features are useful, they cause some complications: NetLuke relies on the operability of the server, users depend on the internet connection.

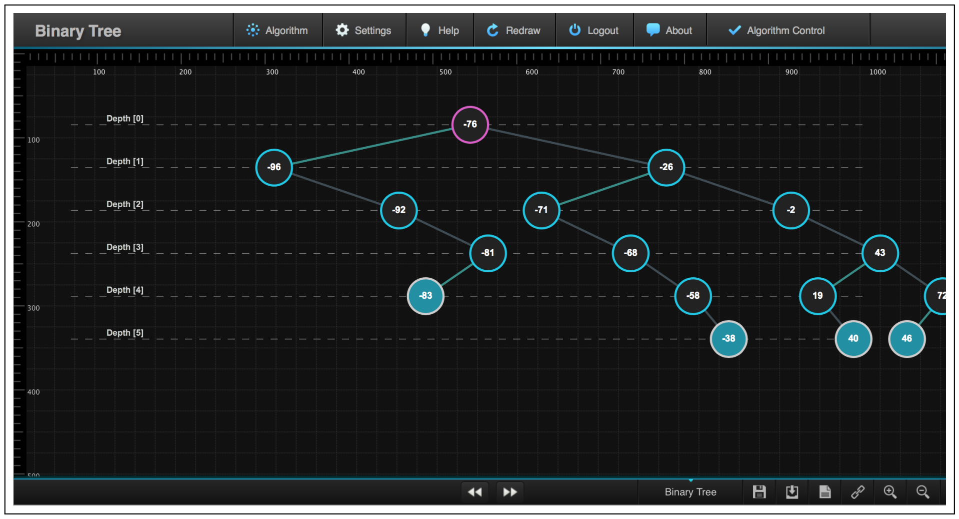
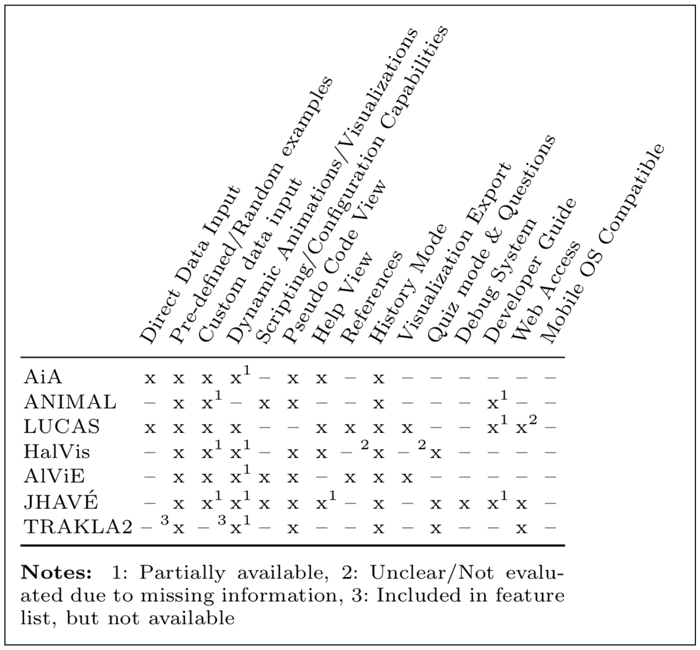


Figure 7: NetLuke Interface [5]

The operating window appears after the log-in. The upper panel includes info, settings, help and algorithm control. The lower panel consists of tape-recorder, buttons for session management and zoom. There are only 5 modules implemented, because the focus of the project today lies at framework flexibility.

2.5 Comparison

There are 7 characteristics picked to compare the 4 tools with webAD: amount of implemented modules, multi device support, presence of tape-recorder, additional theoretical information about topics, limitless dynamics of user input (user may input any combination of numbers), existence of pre-defined examples and low dependence on internet connection (application can operate without internet and limits once downloaded, and the download is provided).
These criteria are chosen, because they focus on two main spheres, which need to be considered by an e-learning visualization tool: coverage of didactic material (modules amount, theoretical info, pre-defined examples) and usability (multi device, tape-recorder, limitless dynamics of user input, low dependence on internet). Additionally, some of them reflect the previously described goals set for webAD, and others are inspired by the analysis performed by Georg Prenner (figure 8) in his paper [41] on NetLuke, if found suitable. For example the input is a crucial criteria for the investigation. However, instead of examining direct and custom user input, it can be aggregated into limitless dynamics of user input.



	Direct Data Input	Pre-defined/Random examples	Custom data input	Dynamic Animations/Visualizations	Scripting/Configuration Capabilities	Pseudo Code View	Help View	References	History Mode	Visualization Export	Quiz mode & Questions	Debug System	Developer Guide	Web Access	Mobile OS Compatible
AiA	x	x	x	x^1	–	x	x	–	x	–	–	–	–	–	–
ANIMAL	–	x	x^1	–	x	x	–	–	x	–	–	–	x^1	–	–
LUCAS	x	x	x	x	–	–	x	x	x	x	–	–	x^1	x^2	–
HalVis	–	x	x^1	x^1	–	x	x	–	^{2}x	–	^{2}x	–	–	–	–
AlViE	–	x	x	x^1	x	x	–	x	x	x	–	–	–	–	–
JHAVÉ	–	x	x^1	x^1	x	x	x^1	–	x	–	x	x	x^1	x	–
TRAKLA2	–	^{3}x	–	$^3x^1$	–	x	–	–	x	–	x	–	–	x	–

Notes: 1: Partially available, 2: Unclear/Not evaluated due to missing information, 3: Included in feature list, but not available

Figure 8: Tool Analysis Performed by Georg Prenner [41]

Some features of the matrix were not taken into account, when constructing the examination characteristics, if found irrelevant. For example, as argumented in section *1.4 Goals*, such components as quizzes or separate pseudocode window (though general pseudocode for complex algorithms is present in webAD) violate our minimalistic spirit and general view on visualization systems; a debug system should not be needed, because we believe, that a system has to be developed completely robust, etc.
As shown, the analysis compared more tools according to more concrete features (which are often unified in analysis of this paper), but unfortunately does not include NetLuke in the comparison.
In the examination of this thesis (table 1) if a certain feature is supported, the according cell contains a checkmark (✓). If it is not provided, the cell includes an x-mark (✗). If it is not known, a question mark (?) is placed into the cell.

	modules amount	multi device	tape-recorder	theoretical info	dynamics of input	pre-defined examples	low internet dependence
JHAVÉ	32	✗	✓	✓	✗	✓	✗
Algorithms in Action	33	✗	✓	✓	✓	✓	✗
VADer	10	✗	✓	?	✓	?	✓
NetLuke	4	✓	✓	✓	✓	✓	✗
webAD	14	✓	✓	✓	✓	✓	✓

Table 1: Comparison of Tools

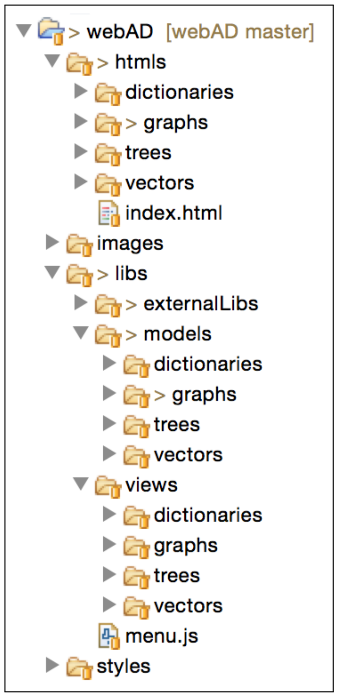
As one can see, webAD supports all the features, which are vital from our point of view. The project includes now only 14 modules, because a single developer has worked on their realization, while all other tools had numerous contributors.
Though all tools have a tape-recorder, they are significantly limited compared to webAD's implementation and have issues. For example, none of them provide switching to the first and last states. NetLuke's realization does not allow the animation to be paused, which makes it very inconsistent when the algorithm is running, and user tries to switch back and forth. Tape-recorder by JHAVÉ is unable to start an automatic animation, to see each single step

one has to click a button every time, which requires an extremely exaggerated interaction effort.
It should be also underlined, that VADer was a project from 1999, when technologies were incredibly limited compared to today.

3 Architecture

3.1 Architecture Overview and File Hierarchy

The skeleton of the system is the Model-View-Controller architectural pattern. Separating the code for application logic from the code for visual representation is of vital necessity for complex applications with the focus on user interfaces. It helps to avoid the big ball of mud.
Models and views are stored hierarchically in file directories. The logical implementations of algorithms and data structures are the models. The views represent subsets of the models and draw them on a HTML5 <div> element. When an instance is created, the model of it is stored in the view object as an instance variable, and, vice versa, the view is stored in its model object as an instance variable. Thus an easy bidirectional access is ensured: a model can trigger the view refreshment when needed, the view can access the model data for drawing. Code snippets in the HTML pages forward user input to models and thus serve as a controller.
From the file hierarchy point of view the architecture is depicted in figure 9. Visible to the user are the HTML pages, located in the *htmls* folder. The classical *index* page serves as the homepage linking everything together. The *images* directory contains all pictures used in the HTML pages in the *Info* section. *libs* directory contains external libraries, which are described further in section *3.4 Tools*. Otherwise it contains models and views. *styles* directory includes a css file styling the HTML pages, a css file with scalable vector graphics for logos and buttons and files for the font. Web Design is performed in cooperation with Begy Kirilo.



```
▼ > webAD [webAD master]
  ▼ > htmls
    ▶ dictionaries
    ▶ > graphs
    ▶ trees
    ▶ vectors
      index.html
  ▶ images
  ▼ > libs
    ▶ > externalLibs
    ▼ > models
      ▶ dictionaries
      ▶ > graphs
      ▶ trees
      ▶ vectors
    ▼ views
      ▶ dictionaries
      ▶ graphs
      ▶ trees
      ▶ vectors
      menu.js
  ▶ styles
```

Figure 9: File Hierarchy

3.2 Thick Client

The two basic approaches for designing the architecture of an application operating over a network are the thin and thick (also called fat) clients. In the case of the thin client, the user's device is used for input and display of information only, while all the calculations are performed by the remote server. This implies a constant connection to the server. Contrary, a fat client does not rely on the connection once it is established, because most of the functionality is loaded on the device.
The argument in favor of the fat client is supported by the nature of this architecture. It is obvious, that it enables a better usability, since the application becomes more responsive and robust once it cannot be interrupted by the loss of the network connection. Consequently the user feels in charge of the application. This is why a thin client should rather be used only when complex computing by server is unavoidable for basic functionality, because such architecture harms the usability a lot.
Thus, a thick client is a better choice for light applications, which do not necessarily require heavy computing, especially if no centrally managed persistence is needed. These are the basic prerequisites for a fat client. A visualization tool for algorithms and data structures meets these requirements, it does barely require hardware resources and no central persistence.
A thick client implies many advantages, which we have as well experienced after realizing the architecture in webAD:

- Improved usability, the functionality cannot be interrupted by network disturbance.
- More responsive design, caused by the absence of roundtrips between server and client for view rendering.
- Primitive server deployment and maintenance.
- Low server costs, since marginal users barely influence the server's workload.

It is also believed, that one of most likely measures for improvement of thin client functionality is the equipping of the client with better resources. Ironically, this will turn a thin client into a good candidate for a thick client [44].

3.3 MVC architectural pattern

Architectural patterns provide rough exemplary solutions for often occurring problems, which arise when designing the software. It has 4 segments: name, problem, solution and consequences. The name's goals are to describe the problem domain shortly and precisely and to build the scientific vocabulary. The problem reflects stereotypical situations and sometimes preconditions for the usage of the according pattern. The solution provides an abstract template, depicting the segments of the design and their relations. The consequences make it clear, what a pattern implementation implies. Though a certain pattern may help to reach a specific goal, the resulting software may have restrictions for other patterns. This factor is important to consider while analyzing the trade-offs and alternatives [38].

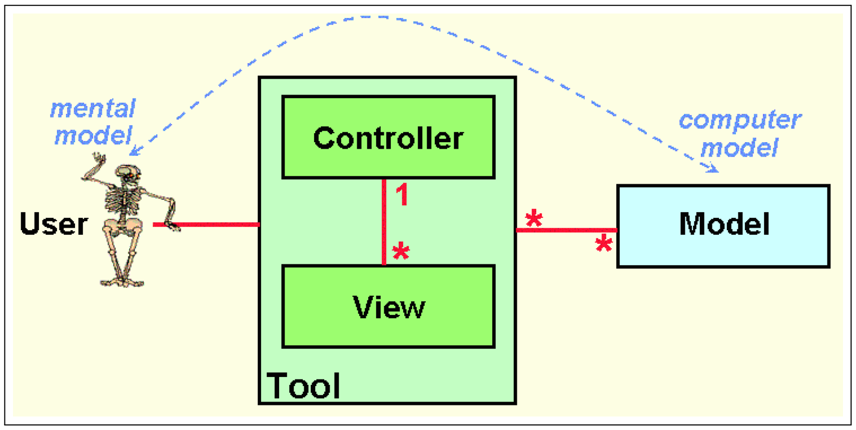


Figure 10: Diagram of MVC by Trygve Reenskaug [11]

Model-View-Controller concept was originally developed by Smalltalk-80 for implementations of user interfaces. Its creator, Trygve Reenskaug, has described the components of MVC and an additional element, the *Editor* in 1797 as follows [42]:

- A Model contains the knowledge of the system. This knowledge is subjective and is defined by the owner of the model. The atomic elements of a well structured model should represent distinctive real world concepts. In webAD typical examples for models are: tree, graph, node, edge, etc.

- A View is a visual representation filter, which highlights the important details of the model and omits the unnecessary ones. All semantic data used by the view needs to be defined by the model. View and model can interact in different ways. In webAD, as mentioned before, model and view are attached to each other at initialization. Though they can access each other bidirectionally at the implementation level, the actual architectural communication occurs in one direction: the model calls the view ones, subsequently the view draws all relevant information without any further requests to the model, only using the information from attached to it model. The procedure is represented graphically further together with a concrete MVC implementation in webAD. Summarizing, the Views in webAD are composites of atomic graphical elements, such as circles or rectangles, which reflect the models.

- A Controller enables the interaction between the user and the application. It parses the input into messages understood by the system. Controller may not by any means invade the view to perform its tasks.

- An Editor is an extension of controller allowing user to update the view directly. It is serving as an intermediate between the view and controller. This feature is not implemented in webAD.

The MVC concept originated long before the use of computers became an ordinary activity. Thus, at its roots it was a simple *"triad of classes"* [38]. Since then, the huge expansion of personal computers has raised the requirements for the software. Mobile devices require developers to use the limited hardware with solicitude. Aimed at end users experience and high-quality implementations, the MVC pattern was destined to become popular and evolve.

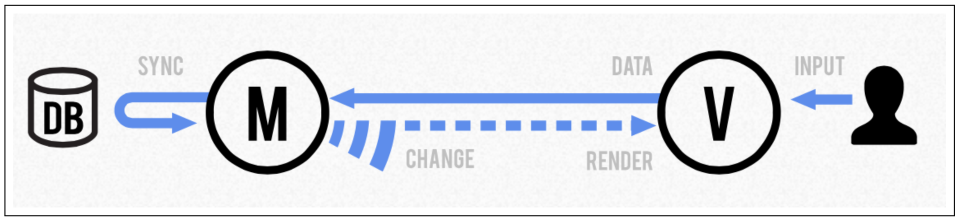


Figure 11: Backbone.js MVC Implementation [9]

First it was developed for desktop applications, but then was quickly adapted for web technologies by numerous frameworks. A famous JavaScript MVC library is Backbone.js, which was originally used by webAD. It provides some handy features, like automatic view rendering and event handlers.
Since the pattern is abstract, model, view and controller are loosely coupled. This lets the architect to determine the final flexible structure. There are many interpretations of MVC with slightly different connections between the core elements. The way the pattern's creator visualized it is depicted in figure 10.
User interfaces are obviously of utter importance in mobile development. Another popular interpretation of the MVC can be found at the Google Chrome documentation for developers:

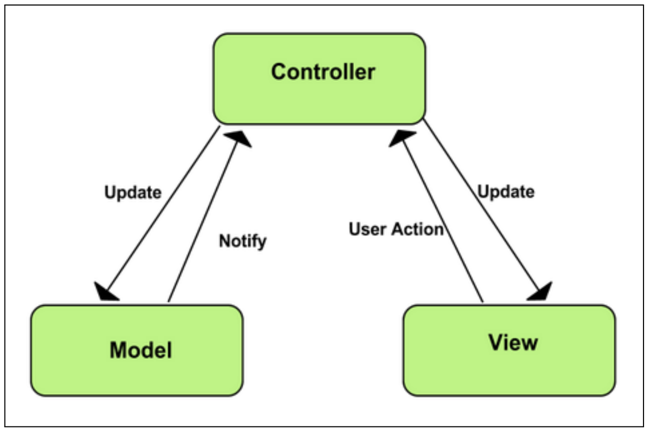


Figure 12: MVC Implementation by Google Chrome [12]

After numerous initial iterations and toggling around with the core design, the MVC version depicted in figure 13 has proven to be most suitable for webAD. This version was chosen because it was perfectly fitting webAD's control flow:

1. User performs the input through the controller in the HTML page.
2. Controller calls the according object-oriented function of the model in the JS file.
3. Model starts to work and updates the view with each step, this way the animation appears.

4. The user sees the resulting views.

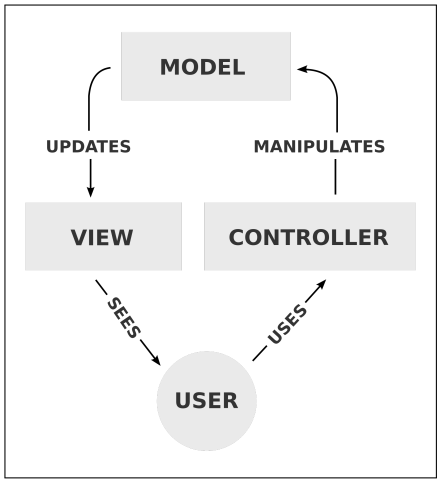


Figure 13: MVC Implemented by webAD [13]

The way the pattern works on more low level is represented in pseudocode:

```
//step 1: initialize model in controller
<script>

  var instance=undefined;

  function init(){
    instance = new Model();
  }

</script>

//step 2: model initialization

function Model(){
  this.view=new View(this);
}

//step 3: triggered view initialization

function View(model){
  this.model=model;
}

```

```
//step 4: model manipulation

Model.prototype.manipulate=function() {
  //manipulate the model here

  this.view.draw(); // call the view

  /*in real code there is a light wrapper for the view's draw
      method in the model's implementation*/
}

//step 5: view update using assigned model

View.prototype.draw=function(){
  //get relevant information for view
  var color=this.model.color;

  //draw...
}
```

Listing 1: MVC on Code Level

3.4 Tools

webAD is built purely on HTML5 and JavaScript. This decision was made for multi-device support, because these technologies run in every popular browser of every operating system on any device. Furthermore they enable the concept of the fat client.

The additional libraries used are KineticJS [6], jQuery [7] and annyang [8]. All of these libraries are pure JavaScript. KineticJS is a HTML5 canvas library for drawing. jQuery is used by controller for handling the user input within the HTML pages in object oriented style with significantly less complex code. annyang is a light wrapper, which allows to write simple code in order to access the built-in browser speech recognition.

Previously two more libraries were used: Backbone.js as a Model-View-Controller implementation and TaffyDB, the JavaScript Database for storing the states for the tape-recorder. They have been dismissed after realization, that only small fragments of their functionality were needed by the system, which could also be programmed more optimally for the specific needs. Instead of storing the states in a TaffyDB instance, they are now stored in a primitive array after being created in a copy constructor.

3.4.1 KineticJS

KineticJS is a de facto standard library for HTML5 drawing. With the last stable release of version v5.1.0 it is not maintained anymore. Countless web applications still use this robust library. KineticJS values the speed of execution and smoothness of animation. Thus developers invest a lot of effort to optimize it, for example providing caching.
It allows to draw and manipulate drawn shapes, providing event handlers. Shapes can also be coordinated in groups, layers and stages. The following figure depicts the usage of the *mouseover* event provided by KineticJS:

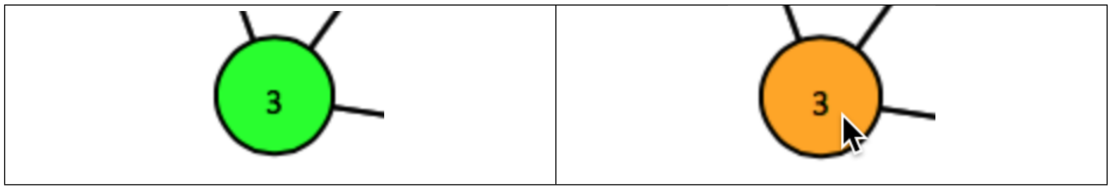


Figure 14: KineticJS *mouseover* Event

3.4.2 jQuery

Used by nearly any modern website, jQuery is a light standard library for handling HTML documents and events performed on HTML elements. In webAD it is used by controller and takes care of code simplicity. Following code excerpts 2 and 3 taken from webAD demonstrate how much the code gets simplified once jQuery is applied:

```
var c1 = document.getElementById("container1");
```

Listing 2: Getting a DOM Element Using Pure JavaScript

```
var c1 = $('#container1')[0];
```

Listing 3: Getting a DOM Element Using jQuery

3.4.3 annyang

Developed by Tal Ater under MIT licence, annyang is a small JavaScript wrapper with 631 LOC for built in browser speech recognition. It allows the

developer to easily map voice commands, which are going to be recognized by the browser and routed to classical JavaScript functions.
In webAD it is used in controller of Bubble Sort. Following listing shows a code snippet, with a spoken command *"sort"* mapped to a JavaScript function *vsort()* using the library:

```
if (annyang) {
  var commands = {
    'sort': vsort
  };
  annyang.addCommands(commands);
  annyang.start();
}

function vsort(){
  //implement the method here
}
```

Listing 4: Adding annyang Commands

3.5 Models

Model for each algorithm and data structure is a JavaScript file. It implements the according class in object oriented style, including according methods, such as deletion, insertion, sorting, etc. Each step within those methods which needs to be visualized is executed with a delay, using the JavaScript *setTimeout()* method.

```
Example.prototype.exampleMethod=function(){

  function wrapper(instance){
    setTimeout(function(){
      //do something to the instance
    },1000/*time of delay in milliseconds*/);
  }

  wrapper(this);
}
```

Listing 5: Backbone of the Animation

The time of delay may be adjusted through *Config* and passed to the model by the controller. At the end of each step the model triggers the refreshing of the view by calling its *draw()* method.

Except for methods for the behavior of algorithms/data structures all models contain several special generic methods. The *saveInDB()* method is usually called, when a new state of the instance is created through the manipulation. It creates a copy of the new state using the *copy()* method in a copy constructor style, then stores it in the array with all the states, if it is not identical to the last saved state. This serves for better usability while using tape-recorder, avoiding redundant pressing of buttons, if several following states are identical (this may arise for example when user starts and pauses the manipulation quickly over and over). Furthermore there are four intuitive methods for the tape-recorder: *prev(), next(), firstState(), lastState()*. They switch between states accordingly using array indexes and call the *replaceThis()* method, which overwrites all instance variables of the object used in the HTML page by the controller with the values of the state stored in the history.

3.6 Views

A view is a JavaScript file as well. It takes the main drawing area, the HTML5 <div> and stores it as a KineticJS *stage*. The view's core is the *draw()* method, which typically looks as follows:

```
SampleView.prototype.draw=function(){

  /* adjust width/height according the state of
  the model, the zoom of the view */
  var width = this.model.size() * this.scale + ...
  var height = ... // analog

  this.stage.setWidth(width);
  this.stage.setHeight(height);

  this.stage.removeChildren(); // delete old layer
  var layer = new Kinetic.Layer();

  //iterate the model's elements, draw them
  for(var i=this.model.size()-1;i>-1;i--){
    var rectangle = new Kinetic.Rect({
      /* set coordinates, color and other properties
      according to this.model */
    });

    layer.add(rectangle);
  }

  this.stage.add(layer);

}
```

Listing 6: Example of Drawing

Furthermore a view contains the methods for zooming, which simply adjust the scale from *0.2* to *3*:

```
SampleView.prototype.zoomIn=function(){
  if(this.scale<3)
    this.scale=this.scale+0.1;

  this.draw();
}

SampleView.prototype.zoomOut=function(){
  if(this.scale>0.2)
    this.scale=this.scale-0.1;

  this.draw();
}
```

Listing 7: Zoom

3.7 Controller

The HTML input elements are linked to JavaScript code snippets, which trigger model/view functionality routing the input:

```
<script>

  var instance=new Model();

  function wrapper(){
    clearTimes(); /* resets active timers, makes sure the
        instance can be manipulated without conflicts */
    instance.routeInput();
  }

</script>
...
```

```
<input type="button" value="Wrapper" id="wrapperButton"
    onclick="wrapper();">
```

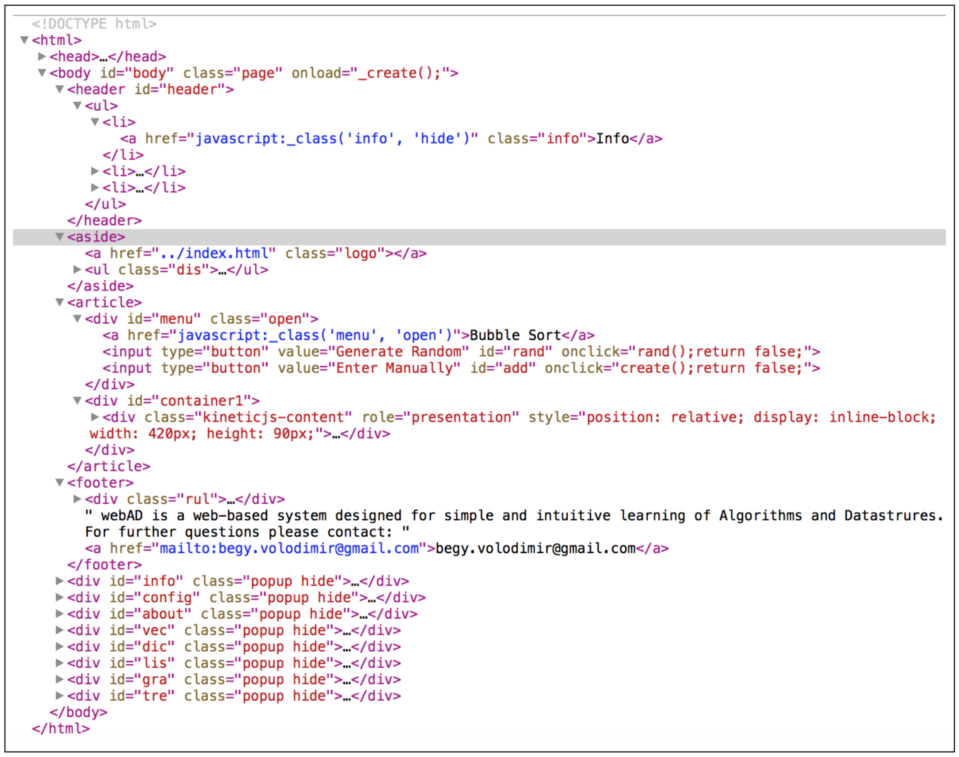
Listing 8: Routing Input

4 Web Design

Web Design is performed in cooperation with Begy Kirilo. For design purposes HTML pages use two css files: a classical *style.css*, which formats the pages and *img.css* with scalable vector graphics in base64 format.

4.1 Structure of the HTML page

Basic building blocks of the HTML pages are HTML <div> tags. They allow grouping of elements and formatting them with the style defined for that <div> tag in the according css file [15].



```
 <!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body id="body" class="page" onload="_create();">
    ▼<header id="header">
      ▼<ul>
        ▼<li>
            <a href="javascript:_class('info', 'hide')" class="info">Info</a>
          </li>
        ▶<li>…</li>
        ▶<li>…</li>
        </ul>
      </header>
    ▼<aside>
        <a href="../index.html" class="logo"></a>
      ▶<ul class="dis">…</ul>
      </aside>
    ▼<article>
      ▼<div id="menu" class="open">
          <a href="javascript:_class('menu', 'open')">Bubble Sort</a>
          <input type="button" value="Generate Random" id="rand" onclick="rand();return false;">
          <input type="button" value="Enter Manually" id="add" onclick="create();return false;">
        </div>
      ▼<div id="container1">
        ▶<div class="kineticjs-content" role="presentation" style="position: relative; display: inline-block;
         width: 420px; height: 90px;">…</div>
        </div>
      </article>
    ▼<footer>
      ▶<div class="rul">…</div>
        " webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures.
        For further questions please contact: "
        <a href="mailto:begy.volodimir@gmail.com">begy.volodimir@gmail.com</a>
      </footer>
    ▶<div id="info" class="popup hide">…</div>
    ▶<div id="config" class="popup hide">…</div>
    ▶<div id="about" class="popup hide">…</div>
    ▶<div id="vec" class="popup hide">…</div>
    ▶<div id="dic" class="popup hide">…</div>
    ▶<div id="lis" class="popup hide">…</div>
    ▶<div id="gra" class="popup hide">…</div>
    ▶<div id="tre" class="popup hide">…</div>
    </body>
 </html>
```

Figure 15: Typical HTML Page

The structure of the page is represented in figure 15.
The <head> tag includes meta information about the page, gets the css files, scripts from the libraries and contains the controller.

The main content is located in the <body> element. The <header> contains links, which trigger popups for *Info, Config* and *About*. The <aside> tag consists of the logo, which leads to the homepage and the right panel, with links, which call popups for chapters of the modules (*Vectors, Dictionaries, Lists, Graphs* and *Trees*). The element <article> has the shrinkable menu with buttons, which operate on the model and the <div> element, which is used as container for drawing by KineticJS. The <footer> consists of the tape-recorder (<div> of class *rul*) and contact information. Afterwards all <divs> for all popups are defined.

4.2 A Word on Popups

Popups are realized through nested <div> tags as well. All popups have one single css class (*popup*), whose design is handled in *style.css*.
A <div> of a popup consists of three major parts:

- An inner <div> element of class *closer*, which represents the grey area in the background and hides the popup when clicked.
- A link in the upper right corner of class *close*, which appears as a red button. It hides the parent <div> analog to the *closer*.
- A <div> element with the contents of the popup.

As said, the popup can be closed either by clicking the red button (which is actually a link) or the grey area. Onclick they call the *_class* function of *menu.js* (stored in *libs* directory). A typical popup looks like this:

```
<div id="vec" class="popup hide">
  <div class="closer" onclick="javascript:_class('vec', 'hide')"></div>
  <a href="javascript:_class('vec', 'hide')" class="close">X</a>
  <div class="pole"><h2>Vectors</h2>
    <ul>
      <li><a href="../vectors/BubbleSort.html">Bubble Sort</a></li>
      <li><a href="../vectors/SelectionSort.html">Selection Sort</a></li>
      <li><a href="../vectors/QuickSort.html">Quick Sort</a></li>
    </ul>
  </div>
</div>
```

Listing 9: Popup for *Vectors* Chapter

In case of popups the *_class(id,someClass)* function called onclick by some HTML elements hides or displays them. It takes the id of the <div> tag, which represents the popup and the name of a certain class. If the HTML element already has the passed class, it is removed. Otherwise it is added to the element. The passed class is named *hide*. The function uses jQuery for simple manipulation of the HTML elements. As shown in listing 9, the *hide* class is added by default to a popup.

```
function _class(id, someClass) {
  var elem = $("#" + id + "");
  if ( elem.hasClass(someClass) ) {
    elem.removeClass(someClass);
  } else {
    elem.addClass(someClass);
  }
}
```

Listing 10: Function Handling Popups

The *hide* class itself is defined in *style.css*. It simply does not display the element.

```
.hide {display:none !important}
```

Listing 11: css hide Class

4.3 Scalable Vector Graphics

Developed by World Wide Web Consortium W3C, Scalable Vector Graphics (SVG) define vector based graphics to be used within browsers.
Since the image is described by vectors, it can be endlessly zoomed in without losing quality as shown in figure 16. Due to this huge advantage the format is recommended by the W3C Consortium.
All images for the HTML pages are SVG images. They are drawn in *Inkscape*, an open-source software editor for creation of SVG. Once the images are exported, they are converted to the base64 encoding using the *b64.io* tool [14].

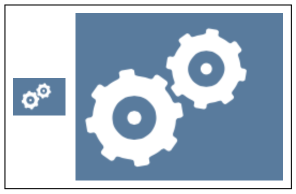
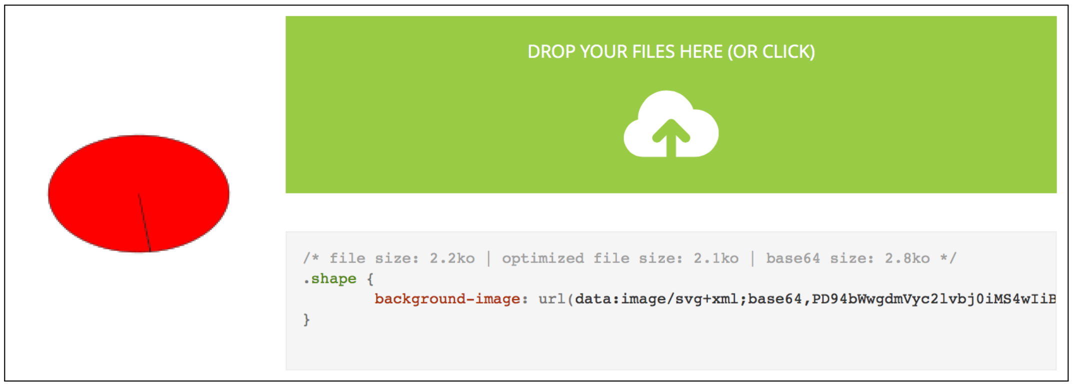


Figure 16: Zooming of an SVG Image



```
/* file size: 2.2ko | optimized file size: 2.1ko | base64 size: 2.8ko */
.shape {
        background-image: url(data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiB
}
```

Figure 17: Simple SVG Shape Converted to base64 [14]

The generated code is stored in the *img.css* file (located in *styles* folder), which is accessed by the HTML pages.
Following pictures within HTML pages are drawn using SVG:

- The logo.
- Pictures representing vectors, dictionaries, lists, graphs and trees in the right panel.
- Buttons of upper panel (*Info, Config* and *About*).
- Buttons of the tape-recorder (figure 18).

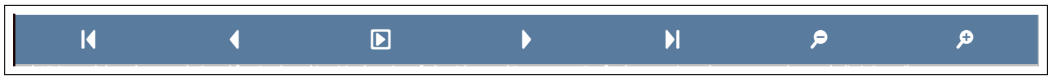


Figure 18: Tape-Recorder Buttons are Drawn as SVG

5 Developer Guide

As of toady, there are 14 modules implemented. The implementation of trivial algorithms and data structures generally follows same patterns, which are described in this section.
Nevertheless, the first step of implementing new modules, especially if they require remarkable control flow (for example as described in section *6.3.3 Graphs*) is brainstorming and trying to fit the new module to the described architecture as closely as possible.

5.1 Creation of the Model

The first step is to create the model in object oriented style with needed methods for the algorithms, which use the previously described timer.
An important factor to consider about execution of methods and algorithms in webAD, is that when user pauses the visualization or operates tape-recorder, instead of using callbacks, all active timers scheduled by the method's execution get aborted. This requires a bit more of implementation effort to design the function consistently: when it is called again, it has to recognize at what exact stage the instance is, skip unnecessary lines of code and jump in at an appropriate line of code. The implementation effort pays off, this design burdens the browser resources less, not forcing it to keep track of callbacks and makes webAD independent from browser specific callback implementations, and code independency is one of major ideologies of the project. The only exception where callbacks are used is Heap Sort, because its tape-recorder is the most unconventional one and callbacks provided the most simple solution for it.
Besides the instance variables responsible purely for the logic, variables representing graphical appearance, such as coordinates or color should be also included and adapted while the algorithm is running. The view should then only draw the object using all information and not calculate it.
Since webAD is a visualization platform, the algorithm and method logic needs to be implemented taking into account several issues: delays of the timer in order to create animation, the fact, that the instance has to be prepared for drawing in the view and the ability to continue running consistently after the function is aborted and then re-entered. Thus, an implementation of a method is always iterative.

- First the developer should program the pure logic, which just works in the background correctly and test it using for example the provided browser debugger.

- Afterwards according code snippets should be wrapped in functions, which are executed with delay. These functions often call each other. For example in Heap Sort first the Max Heap is built with delays for animation. Afterwards independent code for animated sort is called. Thus, using a draft on paper to draw connections between such delayed functions may be helpful.
- Then the created code should be adapted for provision of view-related information, for example setting coordinates of an object in appropriate place.
- Furthermore the call of *saveInDB()* should be placed at the point when a new state arises in execution.
- Now the model should call the drawing of the view at appropriate spots.
- At last the function has to be iterated to work consistently after any abortion by user.

To get a better look at what is meant, compare the two listings 12 (shows the classical implementation of Bubble Sort) and 13 (webAD's Bubble Sort implementation).
The usual Bubble Sort uses a *swapflag* to remember, whether some pair of values was modified during the last iteration. Only if that was the case, it proceeds going from index *0* to *j*, which starts at the end of the array and decreases with each iteration. During one iteration pairs of elements are compared from left to right and swapped if the sorting order is not fulfilled.

```
Vector.prototype.BubbleSort=function(){

  var swapflag;
  var j=this.size();

  do{

    swapflag=false;
    j--;

    for(var i=0;i<j;i++){

      if(this.elements[i+1].value<this.elements[i].value){

        var temp=this.elements[i+1];
        this.elements[i+1]=this.elements[i];
        this.elements[i]=temp;

        swapflag=true;

      }

    }

  }while(swapflag);

}
```

Listing 12: Classical Bubble Sort Implementation

Bubble Sort in webAD is adapted to described requirements and results in a more complex method. First it is checked, whether the vector is of size *1* or is already finished (the finished variable is used to handle abortion of execution by user). If that is the case, colors of the elements are set, vector is redrawn, stored in tape-recorder and the method returns. Otherwise a function *step*, which stands for 1 iteration is called. It immediately sets the colors, stores the state, redraws the vector and defines a delay for the timer, which is declared within it to be *0* if the execution has just started (for smoother animation). Then a function *sort* is triggered, which actually compares elements within one iteration. It sets the delay of outer *step* function, because now when a new iteration will start, it will need extra time to redraw. Then within its timer *sort* checks whether a swap is needed. If that is the case, elements are placed in right order, swapflag is set to *true* and the algorithm notes, that a new state is created, which needs to be stored in tape-recorder. After resetting the colors the saving and then the redrawing occur. The final step in the inner *delay* function decides, which of outer functions needs to be called now using the instance variables:

- *sort*, if the *i* index is smaller than *j* (if the end of iteration is not reached).
- Else, if *swapflag* is *true* and index *j* is greater than *1*, or if *swapflag* is not set, but the array is not sorted (this second clause is used to adapt

to execution abortions by user) *step* is called.

- Otherwise the algorithm finishes and returns.

In cases *1* and *3* the colors are reset, the vector is redrawn and the state is stored. In second case the indexes and the *swapflag* are reset.

```
Vector.prototype.bubbleSort=function(){

  if(this.size()==1 || this.finished){
    this.finished=true;
    this.setColorsBubbleSort();
    this.draw();
    this.saveInDB();
    return;
  }

  function step(vector){
    vector.setColorsBubbleSort();
    vector.saveInDB();
    vector.draw();
    var firstDelay=0;

    setTimeout(function(){

      function sort(vector){
        firstDelay=100*vector.speed;

        setTimeout(function(){
          var extraState=false;
          if(vector.elements[vector.i].value>vector.elements[
              vector.i+1].value){
            extraState=true;
            var temp=vector.elements[vector.i].value;
            vector.elements[vector.i].value=vector.elements[
                vector.i+1].value;
            vector.elements[vector.i+1].value=temp;
            vector.swapflag=true;
          }

          vector.setColorsBubbleSort();
          if(extraState)
            vector.saveInDB();
          vector.draw();

          function delay(vector){
              setTimeout(function(){
                vector.i=vector.i+1;
                if(vector.i<vector.j){
                  vector.setColorsBubbleSort();
                  vector.saveInDB();
                  vector.draw();
                  sort(vector);
                }
                else if((vector.swapflag && vector.j>1)||(!
                   vector.swapflag && !vector.isSorted())){
                  vector.i=0;
                  vector.j=vector.j-1;
                  vector.swapflag=false;
                  step(vector);
                }
                else{
                  vector.finished=true;
                  vector.setColorsBubbleSort();
                  vector.saveInDB();
                  vector.draw();
                }

              },100*vector.speed);
            }
            delay(vector);
          },100*vector.speed);
        }
        sort(vector);
      },firstDelay);
  }
  step(this);
}
```

Listing 13: Bubble Sort Implementation Adapted for webAD

An exemplary implementation is shown step by step in section *5.4 Concrete Implementation Example*.
Furthermore the class should contain the array for the states of the tape-recorder and the counter of the states:

```
function SampleDataStructure(){
  this.view=new AccordingView(this); /* the model is assigned
      in the view's constructor bidirectionally */

  this.db=[]; // the array for the tape-recorder
  this.actStateID=-1; // counter of the actual state

  /* add instance variables for the logic here */

  this.xPosition=0;
  this.yPosition=0;
  /* add other graphical variables here */
}
```

Listing 14: Sample Data Structure

Since there is only one object used by the controller in the HTML page, whose instance variables are replaced by the *replaceThis()* method, an extra method for the creation of new visible to user instance is needed. This method is *init()*, and it just resets all instance variables and may also include some suitable custom functionality, like enabling user to input the variables:

```
SampleDataStructure.prototype.init=function(){
  var order=parseInt(prompt("Order k:")); /* user input */
  if(isNaN(order)||order<1)return; /* user input should
     always be validated */

  this.root=undefined; /* reset instance variables */
  this.order=order;

  this.draw();

  this.saveInDB(); /* store the newly created state */
}
```

Listing 15: init() of Sample Data Structure

The *saveInDB()* function is generic and looks like this:

```
SampleDataStructure.prototype.saveInDB=function(){

  var count=this.db.length-1;

  //if user is not at the last state
  if(count!=this.actStateID){

    //delete all consequent states
    this.db.splice(this.actStateID+1,count-this.actStateID);
  }

  var nextID=this.db.length;

  //create new independent object in copy constructor style
  var new_state = this.copy();

  // get the last state
  var last_id=this.db.length-1;
  var last_state=this.db[last_id];

  var same=true;

  // compare instance variables of the states
  if(last_state==undefined || last_state.elements.length!=
     new_state.elements.length || ... ){
    same=false;
  }

  // store the new state if it is not identical to the last
     one
  if(!same){
    this.db.push(new_state);
    this.actStateID=nextID;
  }

}
```

Listing 16: Method for Storing a New State

Another standard method is *copy()*, which takes the passed instance and returns an independent deep copy of it for storing in the tape-recorder:

```
SampleDataStructure.prototype.copy=function(){

  var newSampleDS=new SampleDataStructure();
  newSampleDS.instanceVar1=this.instanceVar1;
  // copy other instance variables

  return newSampleDS;

}
```

Listing 17: Method for Copying a State

Methods for the tape-recorder can be re-used without modification:

```
SampleDataStructure.prototype.prev=function(){
  if(this.actStateID>0){
    var prev_id=this.actStateID-1;
    this.actStateID=prev_id;
    var rs=this.db[prev_id];
    //make actual state to THIS:
    this.replaceThis(rs);
    this.draw();
  }
}

SampleDataStructure.prototype.next=function(){
  if(this.actStateID<this.db.length-1){
    var next_id=this.actStateID+1;
    this.actStateID=next_id;
    var rs=this.db[next_id];
    //make actual state to THIS:
    this.replaceThis(rs);
    this.draw();
  }
}

SampleDataStructure.prototype.firstState=function(){
  this.actStateID=0;
  var rs=this.db[0];
  //make actual state to THIS:
  this.replaceThis(rs);
  this.draw();
}

SampleDataStructure.prototype.lastState=function(){
  var last_id=this.db.length-1;
  this.actStateID=last_id;
  var rs=this.db[last_id];
  //make actual state to THIS:
  this.replaceThis(rs);
  this.draw();
}
```

Listing 18: Methods of the Tape-Recorder

replaceThis() method used by the tape-recorder is similar to the *copy()* method, it overwrites all relevant instance variables of the instance used by the controller by the values of the instance variables of the state retrieved from the tape-recorder:

```
SampleDataStructure.prototype.replaceThis=function(toCopy){
  this.instanceVar1=toCopy.instanceVar1;
  // copy other instance variables
}
```

Listing 19: replaceThis()

Last typical method is *draw()*, which is a wrapper of the view's according method:

```
SampleDataStrucutre.prototype.draw=function(){
  this.view.draw();
}
```

Listing 20: Model's draw() Wrapper

5.2 Creation of the View

The second step is the creation of the view. A typical view while being constructed (in the model's constructor) gets a model and a scale assigned:

```
function SampleDSView(_model){
  this.model=_model;
  this.scale=1;
}
```

Listing 21: Sample View

The view also contains a method for the stage initialization, which is performed by the controller:

```
SampleDSView.prototype.initStage=function(cont){
  this.stage = new Kinetic.Stage({
    container: cont,
    draggable: true,
    width: 0,
    height: 0
  });
}
```

Listing 22: Stage Initialization

The stage remains constant during the execution. Furthermore the developer can re-use the zoom code described previously and implement the *draw()* method, adjusting the stage dimensions according to the model and replacing the layer.

5.3 Creation of the HTML page with the Controller

The general skeleton of the HTML page can be re-used. The controller goes into the <script> tag on the top of the page. Since it couples the HTML elements with the instance, the usage of jQuery is advised and may reduce the code complexity. This section should include a placeholder for the instance, which is being visualized, so it can be accessed by the controller:

```
var instance = undefined;
```

Listing 23: Placeholder for the Instance

Typically the body triggers the *init()* function when loaded, which initializes the placeholder and its view's stage:

```
<script>
  ...
  function init(){
    instance=new SampleDataStructure();
    var container = $('#container')[0];
    instance.view.initStage(container);
  }
</script>
<body id="body" class="page" onload="init();example();"> ...
```

Listing 24: Initialization

The overall principle of routing within controller is already shown in listing 8. The developer should create functions, which forward all operations (insertion, deletion and others) to model in object oriented style:

```
<script>
  ...
  function modelsOperationX(){
    clearTimes();
    instance.modelsOperationX();
  }
</script>
...
<input type="button" value="Do Something" id="
    modelsOperationX" onclick="modelsOperationX();">
```

Listing 25: Controller Example

5.4 Concrete Implementation Example

This section demonstrates how a selected module from webAD is implemented following general implementation guidelines. To keep it short, the chosen module is Bubble Sort, because the algorithm is simple and some code excerpts from it are already listed.
As said, the first step is to create the model with its generic methods. The two classes used in the module are *Vector* and *Element*. A vector contains an array of elements. The main operable data structure is vector.

```
function Element(value){
  this.color=undefined;
  this.value=value;
}

function Vector(){
  this.view=new VectorView(this);
  this.db=[];
  this.actStateID=-1;
  this.elements=[];
}
```

Listing 26: Structs Used for Bubble Sort

JavaScript is very flexible, developer does not need to define all instance variables, which will be used in different methods in the constructor. It is also possible to initialize the most important and generic ones, others can be added in different methods. On one hand, this has a disadvantage for developers, who are used to high level object oriented programming languages, since that feature is peculiar, and they cannot see immediately, what properties the objects have during execution. On the other hand it spares a lot of redundant code and makes it much more readable, once the developer is used to JavaScript.
The *init()* method looks as follows:

```
Vector.prototype.init=function(){
  this.elements=[];
  this.i=0;this.j=0;
  this.speed=5;
  this.paused=false;this.finished=false;

  this.col1="#00FF80";this.col2="#FF0000";
  this.col3="#F7D358";this.col4="#CC2EFA";

  this.saveInDB();
}
```

Listing 27: init() of Vector for Bubble Sort

The method renews the vector to a fresh state, emptying the elements, resetting the indexes, the speed, variables of tape-recorder and the default colors. At the end the state is stored for tape-recorder.
The stub of the *saveInDB()* method is identical to listing 16, the only snippet worth mentioning is the tunable part, where the new state is compared to the previous one, to check, wheather they are identical or not:

```
Vector.prototype.saveInDB=function(){
...

  if(last_state==undefined || last_state.elements.length!=
      new_state.elements.length || last_state.speed!=new_state
      .speed){
    same=false;
  }
  else{
    for(var i=0;i<new_state.elements.length;i++){
      if(new_state.elements[i].color!=last_state.elements[i].
          color || new_state.elements[i].value!=last_state.
          elements[i].value){
        same=false;
      }
    }
  }

...
}
```

Listing 28: Tunable Part of saveInDB() Method in Bubble Sort

As shown, the first clause checks if the vectors themselves are identical, the second one examines their elements.
The copy method is primitive due to the nature of vector and just requires a simple iteration of all elements:

```
Vector.prototype.copy=function(){
  var newVector=new Vector();
  newVector.finished=this.finished;
  newVector.i=this.i;newVector.j=this.j;

  newVector.col1=this.col1;newVector.col2=this.col2;
  newVector.col3=this.col3;newVector.col4=this.col4;

  newVector.paused=true;
  newVector.swapflag=this.swapflag;
  newVector.speed=this.speed;
  newVector.elements=[];
  for(var i=0;i<this.elements.length;i++){
    newVector.elements.push(new Element(this.elements[i].
        value));
    newVector.elements[i].color=this.elements[i].color;
  }
  return newVector;
}
```

Listing 29: copy() Method in Bubble Sort

If required, developer can set a certain variable to a fixed value here, instead of just copying. For example the variable *paused* is set to *true*, because when a state is going to be retrieved from the tape-recorder, it should be stopped. The four methods of tape-recorder are identicall to the listing 18 and are just wrapped in an if-clause, checking whether the visualization is paused:

```
Vector.prototype.prev=function(){
  if(this.paused){
    if(this.actStateID>0){
      var prev_id=this.actStateID-1;
      this.actStateID=prev_id;
      var rs=this.db[prev_id];
      //make actual state to THIS:
      this.replaceThis(rs);
      this.draw();
    }
  }
  else
    window.alert("Pause the sorting first!");
}

... // three other methods are analog
```

Listing 30: Tape-Recorder of Bubble Sort

The copying within *replaceThis()* method is straightforward, and just like in the *copy()* method the variables for tape-recorder, indexes, colors and elements of the vector are re-assigned:

```
Vector.prototype.replaceThis=function(toCopy){
  this.finished=toCopy.finished;
  this.i=toCopy.i;this.j=toCopy.j;

  this.col1=toCopy.col1;this.col2=toCopy.col2;
  this.col3=toCopy.col3;this.col4=toCopy.col4;

  this.paused=true;
  this.swapflag=toCopy.swapflag;
  this.speed=toCopy.speed;
  this.elements=[];
  for(var i=0;i<toCopy.elements.length;i++){
    this.elements.push(new Element(toCopy.elements[i].value))
       ;
    this.elements[i].color=toCopy.elements[i].color;
  }
}
```

Listing 31: replaceThis() of Bubble Sort

A speciality of vectors is a specific method for setting colors of elements within the array. It makes the code more loosely coupled, it is advised to determine the colors separately. The method sets the colors of the elements using the instance variables of the vector. For example the sorting is finished, all elements are colored gold (default of *col3*). If the sorting is in progress, all elements before index *j* are colored in purple (default of *col4*). Afterwards, the pair of elements, which is being currently compared colored red (default of *col2*) if a swap is needed, otherwise green (default of *col1*).

```
Vector.prototype.setColorsBubbleSort=function(){
  if(!this.finished){
    for(var j=0;j<=this.j;j++){
      this.elements[j].color=this.col4;
    }
    for(var j=this.j;j<this.size()-1;j++){
      this.elements[j+1].color=this.col3;
    }
    if(this.i<this.size()-1){
      if(this.elements[this.i].value>this.elements[this.i+1].
          value){
        this.elements[this.i].color=this.col2;this.elements[
            this.i+1].color=this.col2;
      }
      else{
        this.elements[this.i].color=this.col1;this.elements[
            this.i+1].color=this.col1;
      }
    }
  }
  else{
    for(var i=0;i<this.size();i++){
      this.elements[i].color=this.col3;
    }
  }
}
```

Listing 32: setColorsBubbleSort()

The default colors are set in constructor and may be adjusted by the user in *Config*.
The wrapper for view's drawing is identical to listing 20.
Additionally Bubble Sort has few primitive utility functions, like *setRandomElements(), isSorted(), getElementsByPrompt(), example()* and *size()*.
The view of Bubble Sort is generic and is also used by Selection Sort. It can be generally used by any sorting algorithm applied on vector, which visualizes only its elements (contrary to Quick Sort, which also depicts pointers). This is why it is stored in the file *default.js* (in the directory for vector views).
The constructor, *initStage(), zoomIn()* and *zoomOut()* correspond to listings 21, 22 and 7 accordingly. The *draw()* method simply iterates the vector and draws its elements using pre-defined colors, setting the sizes according to the value of the model and scale of the view. The width of the stage is reflecting the size of the vector, the height is set according to the largest number within array, supporting responsive design.

```
VectorView.prototype.draw=function(){
  var w=this.model.size()*30*this.scale;
  var biggestNum=0; // to define how high the stage should be
  for(var i=0;i<this.model.size();i++){
    if(this.model.elements[i].value>biggestNum)
      biggestNum=this.model.elements[i].value;
  }
  var h=(45+biggestNum*3)*this.scale;
  this.stage.setHeight(h);this.stage.setWidth(w);
  this.stage.removeChildren();
  var layer = new Kinetic.Layer();

  for(var i=this.model.size()-1;i>-1;i--){
    var rect = new Kinetic.Rect({
      x: 10+(i*25*this.scale),
      y: 20*this.scale,
      width: 15*this.scale,
      height: (5+this.model.elements[i].value*3)*this.scale,
      stroke: this.model.elements[i].color,
      fill: this.model.elements[i].color,
      strokeWidth: 2*this.scale
    });
    var text = new Kinetic.Text({
      x: rect.getX()-8*this.scale,
      y: rect.getY()-15*this.scale,
      text: this.model.elements[i].value,
      fontSize: 15*this.scale,
      fontFamily: 'Calibri',
      fill: 'black',
      width: 30*this.scale,
      align: 'center'
    });
    layer.add(rect);
    layer.add(text);
  }
  this.stage.add(layer);
}
```

Listing 33: Drawing Bubble Sort and Other Trivial Sorting Algorithms

Just like any other controller, controller of Bubble Sort just passes the user input from the shrinkable menu with operations, tape-recorder and *Config* to the model. For example following method resets the first color of the legend according to the color picker from *Config* and applies the changes on the view and tape-recorder. External library jQuery is used to interact with HTML elements in a simple manner. Other controller functions are implemented in an analog way. The speciality of Bubble Sort, is that its controller also handles user input through speech using the annyang library.

```
<script>
  ...
  function col1(){
      if(vec!=undefined && vec.size()>0){
        var col1 = $("#col1");
        vec.col1=col1.val();
        vec.setColorsBubbleSort();
        vec.draw();
        vec.saveInDB();
      }
      else{
        alert("No vector created!");
        resetFields();
      }
  }
</script>
...
<input type="color" name="col1" id="col1" onchange="col1();
   return false;">
```

Listing 34: Sample Method from Controller of Bubble Sort

Table 2 lists all functions of Bubble Sort controller and describes their tasks, to give a better overview of a typical controller content.

Name of the Function	Description of Functionality
vsort()	Enables triggering of the sorting by user's speech input
vstop()	Stops the sorting using speech input
vslow()	Resets the speed of the sorting to the minimum (10) using speech input
vfast()	Resets the speed of the sorting to the maximum (1) using speech input
_create()	Initializes vector onload of the HTML page
resetFields()	Synchronizes the input fields of the HTML page with the variables within the model
speed()	Updates the speed of sorting in the model after user input
col1() ... col4()	Updates the default colors from 1 to 4 for vector's elements in the model after user input
clearTimes() [30]	Aborts all active and scheduled timers
create()	Sets up a new vector with manual user input of the elements
rand()	Sets up a new vector with random elements
pre()	Forwards input from tape-recorder's *previous* button to the model
next()	Forwards input from tape-recorder's *next* button to the model

first()	Forwards input from tape-recorder's *first state* button to the model
last()	Forwards input from tape-recorder's *last state* button to the model
un_pause()	Pauses the visualization if it is running and resumes it, if it is paused
zoomIn()	Uses the zoom in of the view
zoomOut()	Uses the zoom out of the view

Table 2: Functions of Bubble Sort's Controller

5.5 Other Remarks

The experience has shown, that following can be also helpful:

- When implementing the visualization of a sorting algorithm on a vector, as shown in the concrete implementation example of section *5.4* the developer should create a method for setting colors of vector's elements according to the state.

- When the data structure is too complex to traverse for the deep copy purposes, the developer may use a history variable to keep track in what order elements have been manipulated in order to reproduce the instance.

- The developer should become familiar with the used external libraries before beginning the implementation: jQuery, KineticJS.

- webAD uses annyang library as a wrapper for writing commands for built in browser speech recognition. An example is implemented in controller of Bubble Sort.

6 User Guide

6.1 Administrator Guide

Due to the architecture of the platform no complex deployment is needed to make webAD available. To get an own copy the administrator should follow these steps:

- Download the project from the GitHub repository.
- Provide the access to the *index.html* page.

The project can be downloaded at following url either by clicking *Clone in Desktop* or *Download ZIP*:
`https://github.com/VolodimirBegy/webAD`
In first case the GitHub Desktop software needs to be pre-installed. In second case the archive is simply downloaded and extracted.
Another option to download the source is terminal based. Administrator needs to type following command:

```
git clone https://github.com/VolodimirBegy/webAD.git
```

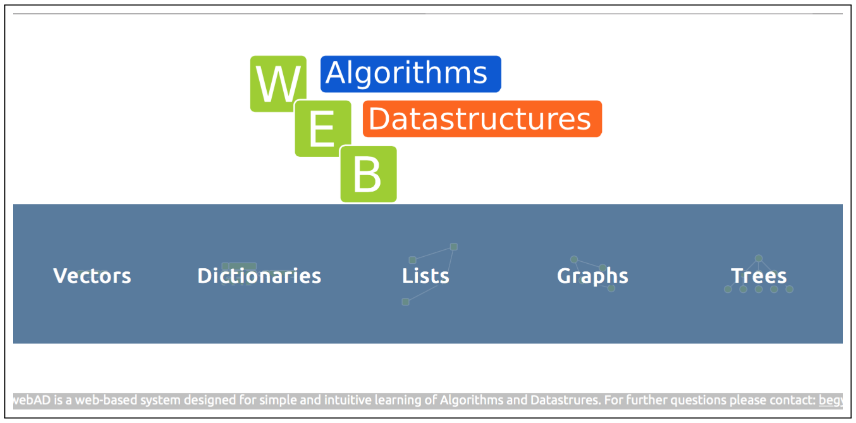
Listing 35: Terminal Based Download

The platform can be then easily used locally by opening the *index.html* in the *htmls* directory.
To provide an online access the folder needs to be placed on the server, and the project is then accessible through an analog link:
`http://administratorsDomain.com/webAD/htmls/index.html`

6.2 User Tutorial

6.2.1 Overall Structure

The homepage contains all algorithms and data structures sorted into topics according to didactic structure of the lecture (*Vectors, Dictionaries, Lists, Graphs* and *Trees*) and the logo. The homepage can be accessed from any page by clicking on this logo.
User interface is adapting to browser resizing and zooming on any device. Combined with the previously described Scalable Vector Graphics this serves for a better look & feel experience.



webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: beg

Figure 19: Homepage

When a certain topic is clicked, a popup with all assigned algorithms and data structures appears:

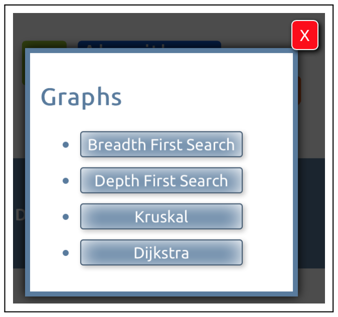
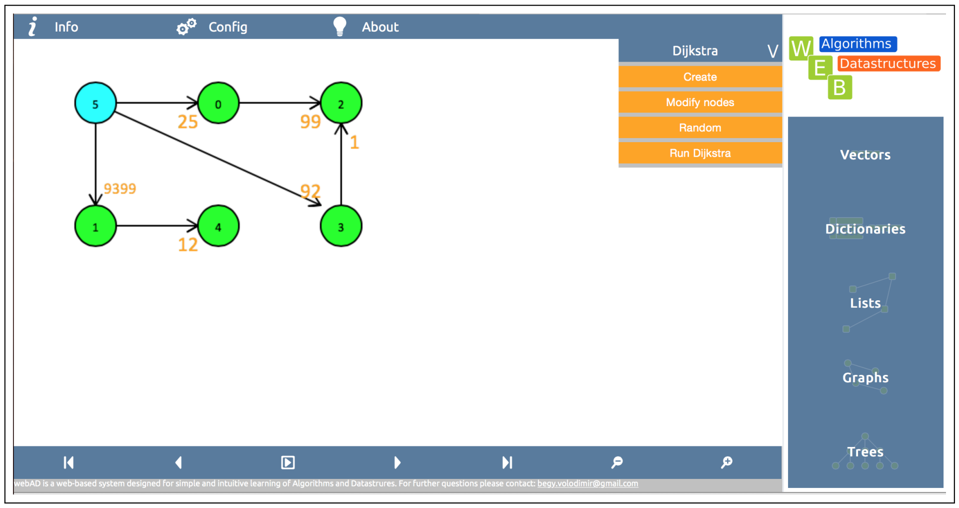


Figure 20: Popup

6.2.2 Algorithm/Data Structure Manipulation Window

When a concrete algorithm/data structure is chosen, the main operating area appears. It consists of the large <div> element where the drawing occurs in the center, control panel on top with *Info, Config, About* buttons and a menu with buttons for the algorithm control, which un-shrinks when clicked. In the right upper corner is the logo, which, when clicked leads to the homepage.
The right panel contains all topics, which trigger the popups with links.
The lower panel contains buttons of the tape-recorder and zoom.

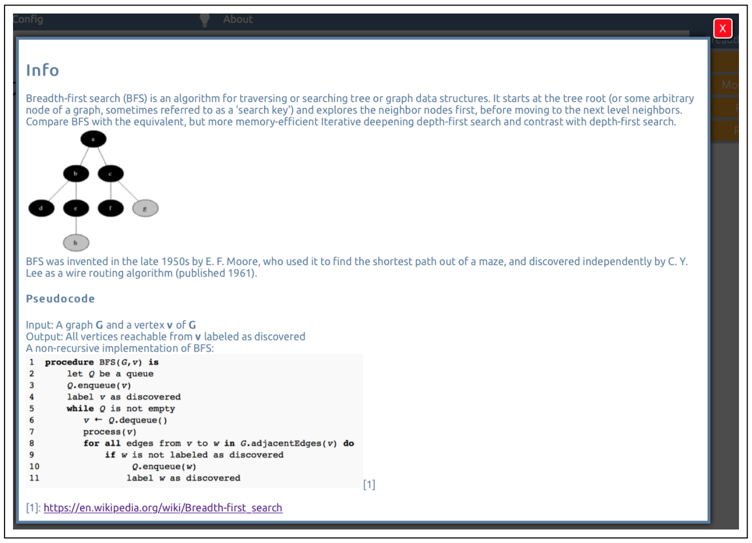
The operating area shows a hardcoded example by default.



webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gmail.com

Figure 21: Operating Area

The *Info* section contains description of the algorithm/data structure with legends, theory, pictures, animations and pseudocode.



Info

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key') and explores the neighbor nodes first, before moving to the next level neighbors. Compare BFS with the equivalent, but more memory-efficient Iterative deepening depth-first search and contrast with depth-first search.

BFS was invented in the late 1950s by E. F. Moore, who used it to find the shortest path out of a maze, and discovered independently by C. Y. Lee as a wire routing algorithm (published 1961).

Pseudocode

Input: A graph **G** and a vertex **v** of **G**
Output: All vertices reachable from **v** labeled as discovered
A non-recursive implementation of BFS:

```
procedure BFS(G,v) is
    let Q be a queue
    Q.enqueue(v)
    label v as discovered
    while Q is not empty
       v ← Q.dequeue()
       process(v)
       for all edges from v to w in G.adjacentEdges(v) do
           if w is not labeled as discovered
                Q.enqueue(w)
               label w as discovered
```
[1]

[1]: https://en.wikipedia.org/wiki/Breadth-first_search

Figure 22: Info Example [22]

The *Config* section displays settings specifically for the according algorithm/data structure. For example for a vector the sorting speed and the color may be adjusted, for Double Hashing/Linear Probing the user can define whether the table is extendible, for Breadth/Depth First Search the graph can be chosen to be un-directed.

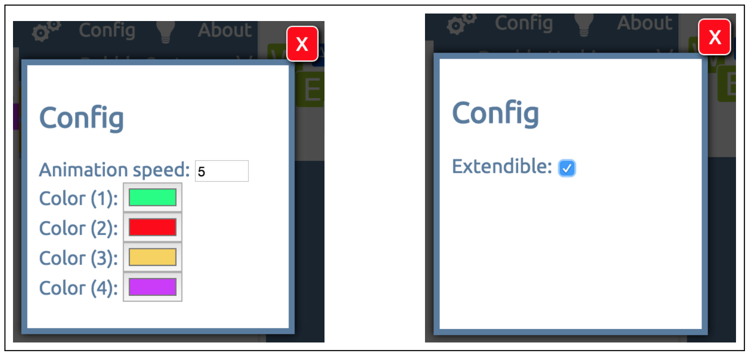


Figure 23: Some Configs

The *About* section has relevant links, such as the University and GitHub repositories.

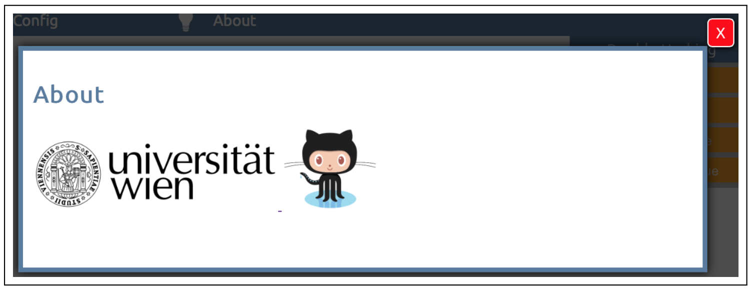


Figure 24: About [4]

Graphs contain an additional component of the interface, the mini adjacency matrix, which is appended below the panel with operations. It is provided, because graphs, which have different models may appear identical in the operating window. This may be caused by disconnection of nodes from the path, which initiates from the first node. When such instances are switched in the tape-recorder, without the additional matrix user may get the feeling, that nothing is changing. To avoid the described issue the mini matrix is introduced.

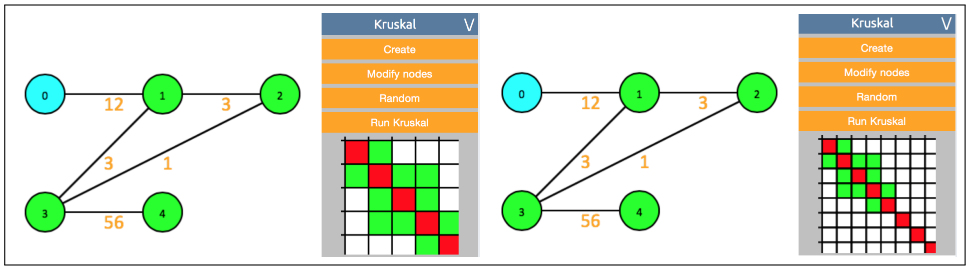


Figure 25: Mini Matrix

6.3 Implemented Topics

At present there are 14 implemented topics:

- Vectors
  - Bubble Sort
  - Selection Sort
  - Quick Sort
- Graphs
  - Breadth First Search
  - Depth First Search
  - Kruskal Algorithm
  - Dijkstra Algorithm
- Dictionaries
  - Double Hashing
  - Linear Probing
  - Linear Hashing
- Trees
  - Binary Search Tree
  - B+ Tree
  - Min Heap
  - Heap Sort

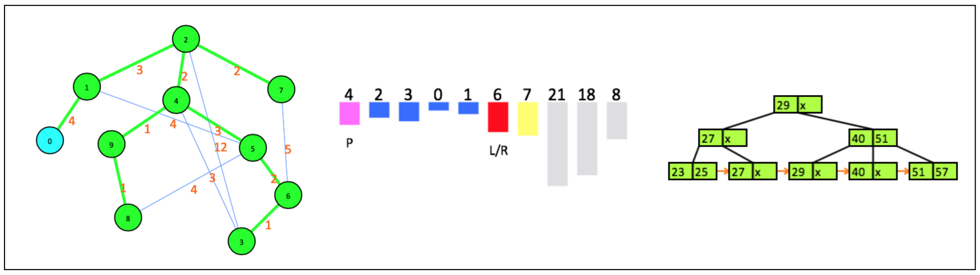


Figure 26: Some Implemented Topics

These 14 modules were chosen in accordance to their relevance to the *Al-*

gorithms and Data Structures lecture: the most important ones were implemented with higher priority.
Manipulation of all the topics is very intuitive, all of them support a tape-recorder and zooming. Trivially tape-recorder is capable of switching between multiple instances, the only exceptions are Breadth/Depth First Search, Double Hashing and Linear Probing, when the class of instance is changed. If some module has a special feature while using tape-recorder, it is described in dedicated to it section.

6.3.1 Vectors

Terminology
Vector is a data structure storing a predefined amount of homogeneous elements. These elements are accessed by the index.
Sorting algorithms are most often applied on vectors.
In webAD sizes of vector's elements are corresponding to the numbers.

Bubble Sort
Bubble Sort is the most primitive sorting algorithm. Given a vector of n elements the algorithm classically performs m=n steps. During each step it goes from first to (n-m)-th element and compares pairs of elements on its way, swapping them if the first element within the pair is greater than the second one in the ascending mode and conversely in the descending mode. In modified version the algorithm is slightly sped up using the so called swapflag: an introduced variable notes, whether some pair of elements was swapped during the last run. If that was not the case, in the next iteration the algorithm knows, that the array is already sorted and finishes. webAD's Bubble Sort implementation uses the swapflag principle.

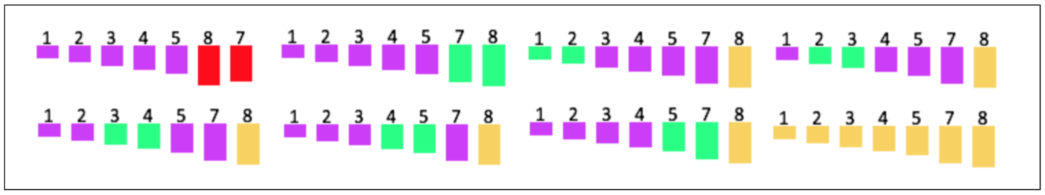


Figure 27: Swapflag

The simplicity pays off through high complexity of $O(n^2)$.

In webAD the ascending mode of Bubble Sort is implemented.

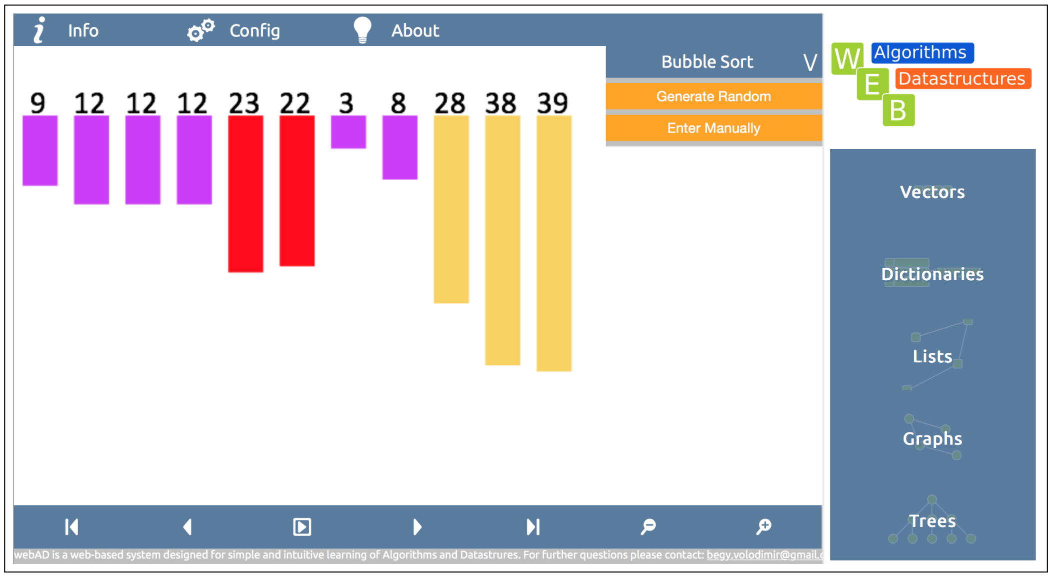
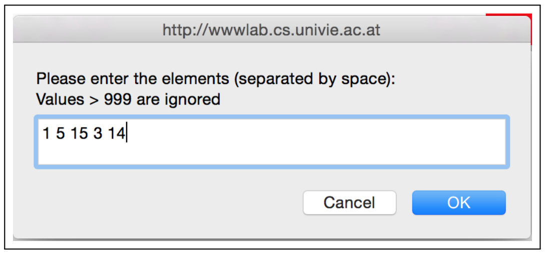

Info
Config
About
Bubble Sort
Generate Random
Enter Manually
9 12 12 12 23 22 3 8 28 38 39
Algorithms
Datastructures
Vectors
Dictionaries
Lists
Graphs
Trees
webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gmail.

Figure 28: Bubble Sort

The pair of elements, which is being compared is marked green, if they do not need to be swapped. If a swap is needed, they appear red. Already sorted values are drawn gold, and all other elements are purple. This default color legend can be adjusted in *Config*.
The animation speed can be chosen between 1 (the fastest) and 10 (the slowest), while default is 5.


http://wwwlab.cs.univie.ac.at

Please enter the elements (separated by space):
Values > 999 are ignored

1 5 15 3 14

Figure 29: Manual Input of Vector

Bubble Sort supports following operations:

- Generate Random: creates a vector of random size with random elements.

- Enter Manually (figure 29): user may enter vector's elements separated by space. Unsupported symbols will be ignored (for example input *1 test 5 ! 3* will result in a vector *1 5 3*).

In order to use the tape-recorder, the algorithm needs to be paused. It can be launched again from any chosen state.
Bubble Sort supports speech recognition of several commands:

- *Sort*, which starts or resumes the sorting procedure. If the command is launched while the algorithm is already running, the browser responds *"Already sorting!"*.

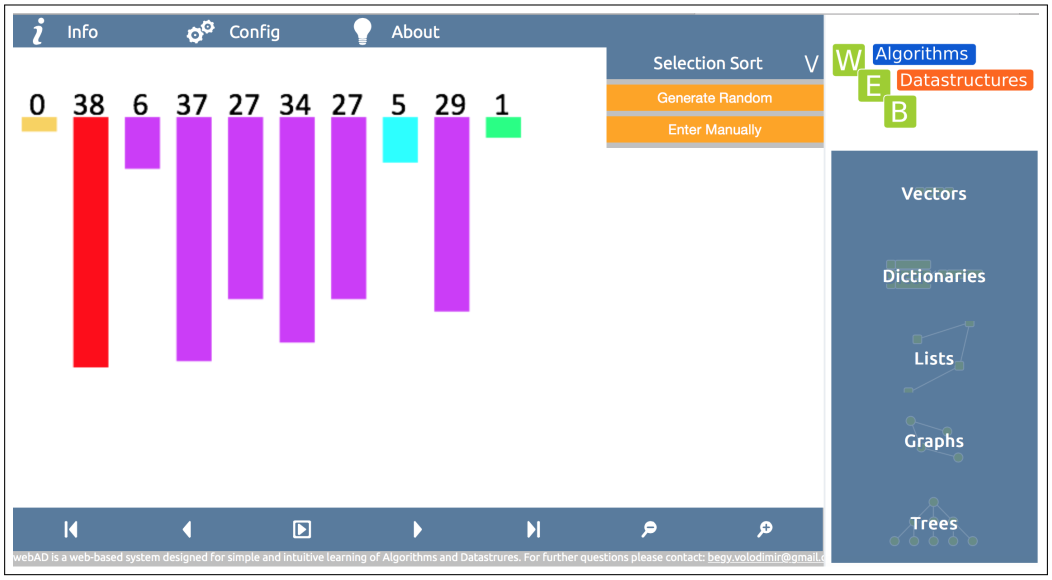
- *Stop*, which pauses the sorting procedure. If the command is pronounced when the algorithm is inactive, the browser responds *"Already stopped!"*.

- *Faster*, which sets the speed of sorting to the fastest possible (1). The changes of speed are synchronized with *Config*.

- *Slower*, resetting the animation speed to the slowest one (10). The changes appear in *Config* as well.

The usage of this feature requires the user's browser to contain built-in speech recognition.

Selection Sort
Selection Sort is another simple, but not very efficient sorting algorithm. Just like Bubble Sort, while working on a vector of n elements it takes m=n steps. However, it starts at the m-th element and proceeds to the last one. On its way the algorithm memorizes the index of the smallest element of the iteration. If that element is not placed at the m-th place (the beginning of the iteration), it is swapped with the m-th element. The described procedure is valid for ascending mode, opposite applies for descending sorting.
Selection Sort is as well of complexity $O(n^2)$.
webAD provides the ascending implementation of Selection Sort.
The current element of the iteration is drawn green. The smallest element of the iteration is marked blue. If the m-th element is going to be swapped, it is appearing red, otherwise, if it is already the smallest one, it appears blue. Elements, which are already sorted are in gold color. Other elements are purple. These colors can be modified in the *Config* menu.



webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gmail.c

Figure 30: Selection Sort

The animation speed is also tunable. Since the algorithm is a bit more complex than Bubble Sort and requires more cognitive effort while watching, the speed interval is twice as wide: default is 10, fastest is 1, slowest is 20. Just like Bubble Sort, Selection Sort supports two operations:

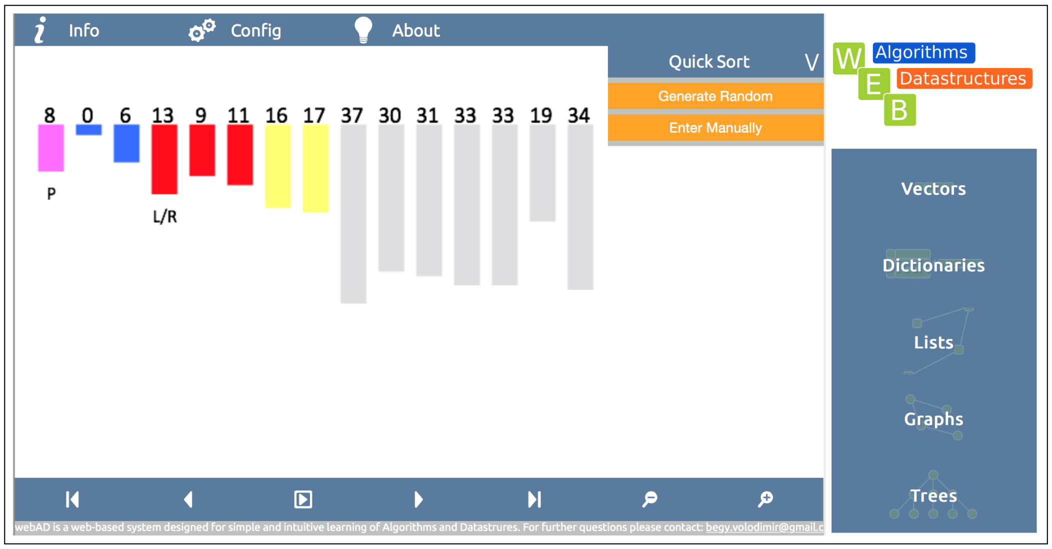
- Generate Random.
- Enter Manually.

The usage of tape-recorder requires the algorithm to be paused. It can be started again from any state.

Quick Sort

Along with Merge Sort and Heap Sort, Quick Sort is a wide-spread efficient sorting algorithm. In practice it is often implemented recursively. At first the procedure selects the pivot (most right or most left element of the sorting range) and sets a pointer on most left and most right elements from the current partition. Then the left pointer begins to wander right unless it finds an element greater than pivot or reaches the edge of partition. Once that event occurs, the right pointer begins to move left in an analog way, unless it finds an element smaller than pivot or meets the end. A very important

factor to take into account while implementing the algorithm, is that either the left or right pointer needs to stop at an element, which may be equal to the pivot element. Otherwise, with a suitable array of numbers, the sorting may fail. When both pointers find the desired elements, they swap them and continue to wander. This process occurs until the path of left and right pointers has crossed. Once they have intersected, the right pointer goes left to find an element smaller than pivot for the last time. When this element is found at index i, it is swapped with the pivot element. The pivot, which is now at the position i is sorted at its final index. The described procedure is then applied to both left and right partitions, which emerge on the sides of the currently sorted element. This way the field is sorted in ascending order. The reverse manner implies descending order.
In average Quick Sort has a complexity of $O(n \log n)$.
In webAD Quick Sort with ascending order and most left element as pivot is visualized.



webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gmail.c

Figure 31: Quick Sort

The pivot element is colored pink. Elements within the current partition, which are smaller than pivot element are marked blue. The ones greater than pivot are red. Elements equal to pivot are also pink. Sorted ranges are depicted in yellow, and ranges, which are going to be processed later are grey. These default colors are tunable through the *Config*. The actual index of pointers is reflected through the letter P (pivot) / L (left) / R (right). Pointers may intersect and be placed on same elements:

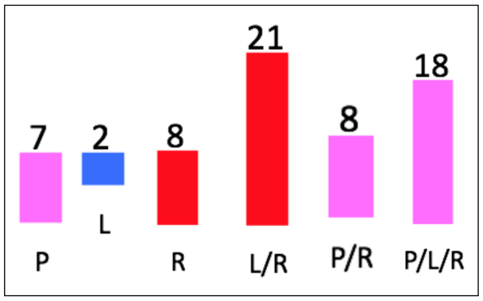


Figure 32: Pointers

Animation speed can be defined by the user in *Config*. The algorithm is complex, and thus the available speed ranges from 1 (the fastest) to 20 (the slowest) with a chosen default of 10.
Quick Sort provides two same operations, like other vector algorithms:

- Generate Random.
- Enter Manually.

Tape-recorder runs analog to Bubble and Selection Sort: precondition for it is that the algorithm is paused.

6.3.2 Dictionaries

Terminology

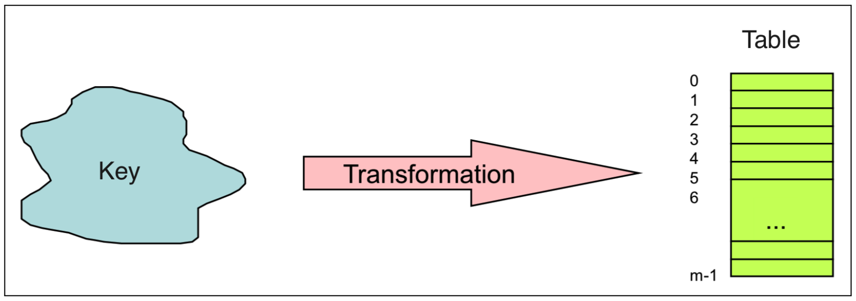
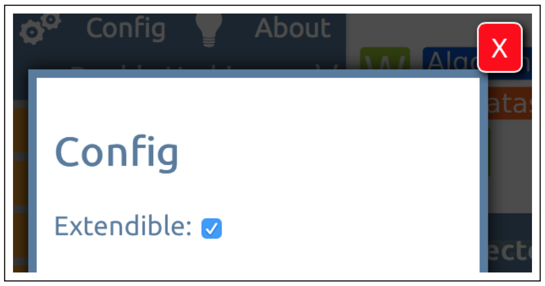


Figure 33: Hashing [39]

A dictionary is a data structure managing entries, which represent key-value pairs. The key provides a mechanism for the access of the entry, while the value stores the actual information. A basic approach to locate the entry is hashing, which determines the index by mathematical transformation of the key. The transformation is performed by the hash function.
Different hashing arts arise with respect to the type of the dictionary itself (static/dynamic) or the way collisions are handled.

Double Hashing

Double Hashing is applied to static hash tables, which means they are not extended when the fill factor is exceeded or there is no place for new entries. It calculates the index for the entry using first hash function. If that function does not succeed (for example in case of search or insertion the entry is occupied by other value), it appeals to the alternative hash function, which is usually based on the first one.
In webAD the first hash function is *k mod m* (*m* stands for the amount of entries). The alternative hash function is *g(k)= R- (k mod R)*, while *R* is the first prime number smaller or equal *(m-1)*.
The module has two modes: the static, which is selected by default, and the extendible, which may be turned on by the user in the *Config*.

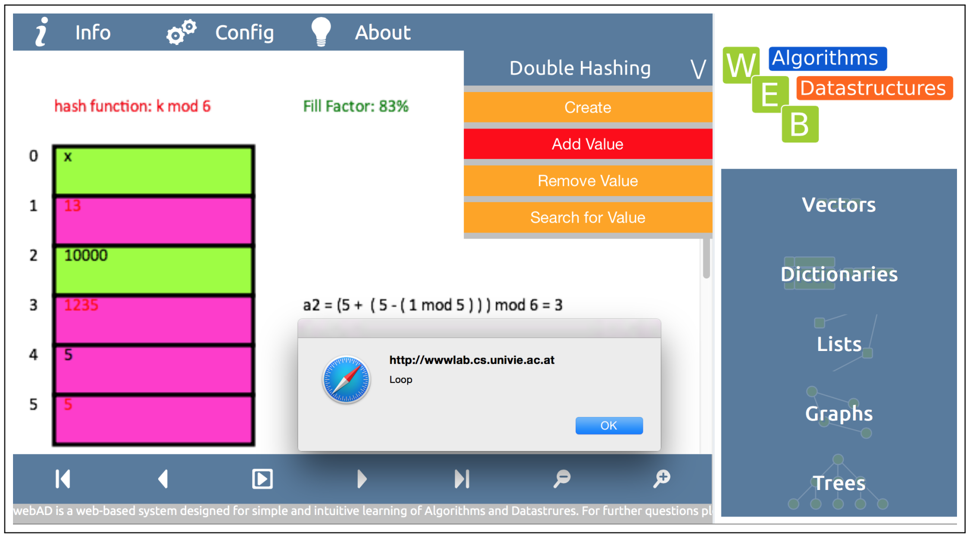


Config

Extendible: ✓

Figure 34: Config of Double Hashing

In the static mode the user has the freedom to create the table of any size. Given the two hash functions, this implies the dangers of a loop, when the table size is not of a prime number. As a result a number cannot be added, even though there is still free space within the table. Figure 35 depicts such situation with described hash functions, a table of size *6*, already stored values (*13, 10000, 1235, 5* and *5*), when *1* needs to be added. After the message the loop gets aborted and information within the entries on its path is marked red (not to be confused with entries marked purple, when included in the collision path).



hash function: k mod 6

Fill Factor: 83%

0	x
1	13
2	10000
3	1235
4	5
5	5

a2 = (5 + (5 - (1 mod 5))) mod 6 = 3

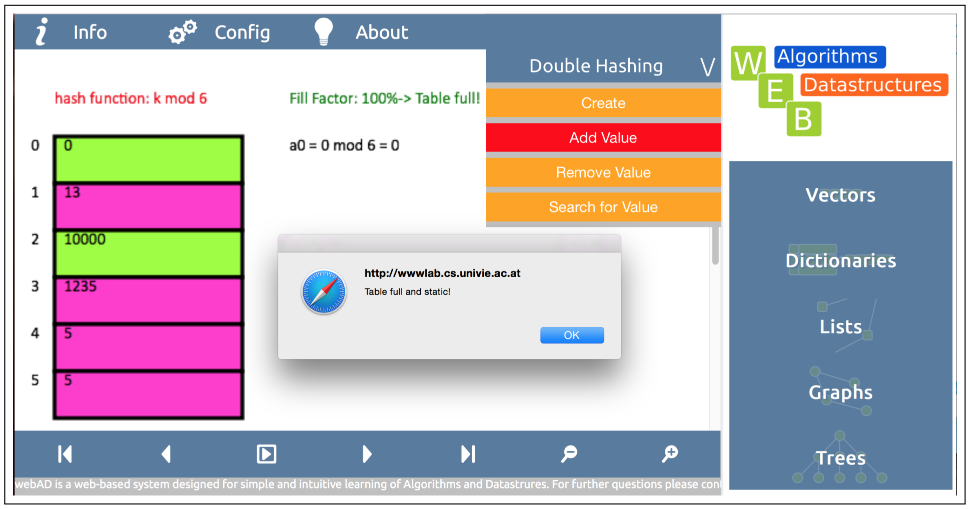
http://wwwlab.cs.univie.ac.at

Loop

webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions pl

Figure 35: Loop in Static Double Hashing

When the table is filled fully in this mode, further insertion is not possible (figure 36).



hash function: k mod 6

Fill Factor: 100%-> Table full!

a0 = 0 mod 6 = 0

0	0
1	13
2	10000
3	1235
4	5
5	5

http://wwwlab.cs.univie.ac.at

Table full and static!

webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please con

Figure 36: Full Table in Static Double Hashing

In extendible mode the size of table is specified automatically. Starting at $m=2$, when the table reaches the fill factor of 70%, it always extends to the size of next prime number and rehashes existing values. Using this approach and described hash functions a loop is impossible and values can be inserted without limits. The rehashing is fully animated, the values in the queue appear under the hash function (figure 37).

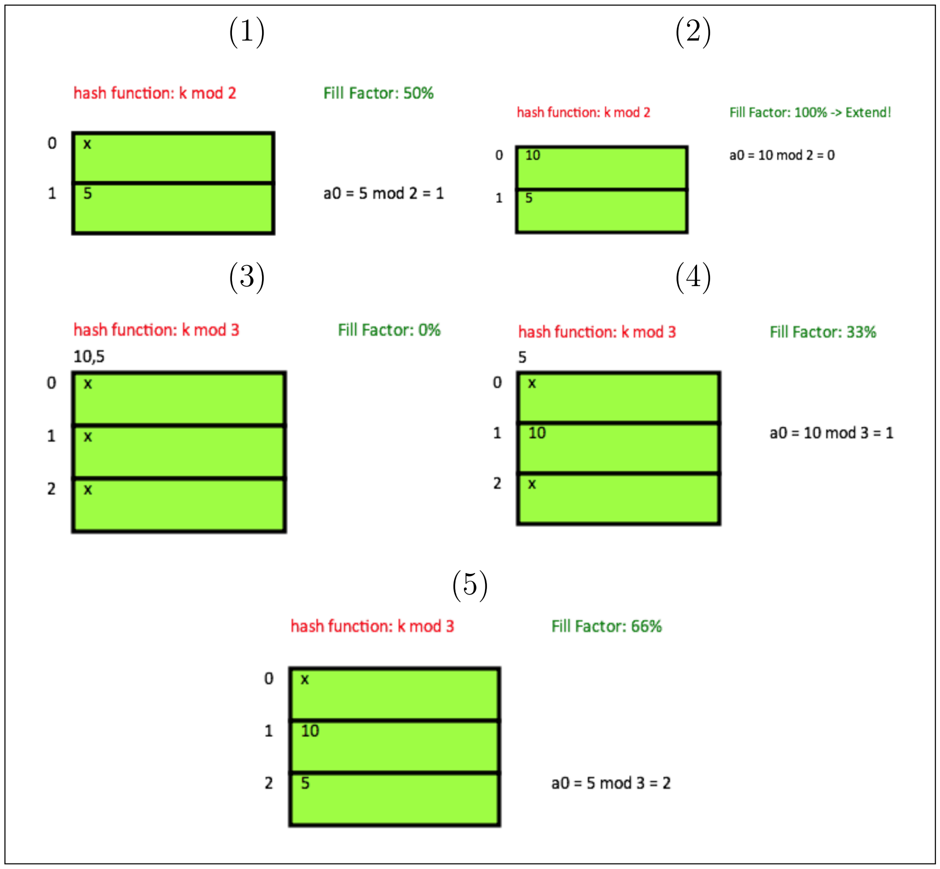


Figure 37: Extension and Rehashing

As already shown on the figures, when a certain operation is performed on the table, the calculation of the index is written next to the calculated entry. Entries which were deleted or collided on insertion are marked purple, they need to be included in the collision path: while searching for a value and hitting on an empty purple entry, the algorithm may not assume, that the search is over and has to continue.

Double Hashing provides following operations in both modes (static/extendible):

- Create, which sets up a table of size determined by user in static mode and of size *m=2* in extendible mode.
- Add Value, inserting a new value if no loop occurs and the table is not full in static mode and without limits in extendible mode. Insertion in extendible mode may trigger table extension and rehashing of all values, if the fill factor exceeds 70%.
- Remove Value, deleting information from the entry and including it in the collision path.
- Search for Value, looking up information, eventually using the collision path.

Tape-recorder may be used any time when the table extension is not in progress.

Linear Probing
Linear Probing is a special type of Double Hashing. Its second hash function, which is used for collision handling searches for the next free space in linear manner: $index = (p + 1) \bmod m$, where p is the previously investigated index and m is the table size. Due to the nature of this alternative hash function, no matter what table size the user chooses, a loop may not be triggered, because the algorithm is always looking at the next entry.
The simplicity of the procedure pays off in lack of performance, it is not optimal to search linearly for the next available slot.
Given a table of size 4, which already contains values *1, 2, 3* the working principle of the alternative hash function is depicted in figure 38 by inserting *1*.
Just like in Double Hashing, webAD contains two modes of Linear Probing: static by default and configurable extendible. As mentioned before, in static mode no loop can occur. When the table is full, no new value can be inserted and a message analog to figure 36 appears. In extendible mode the fill factor is 70%.
The color legend is identical to Double Hashing: entries from the collision path are purple.

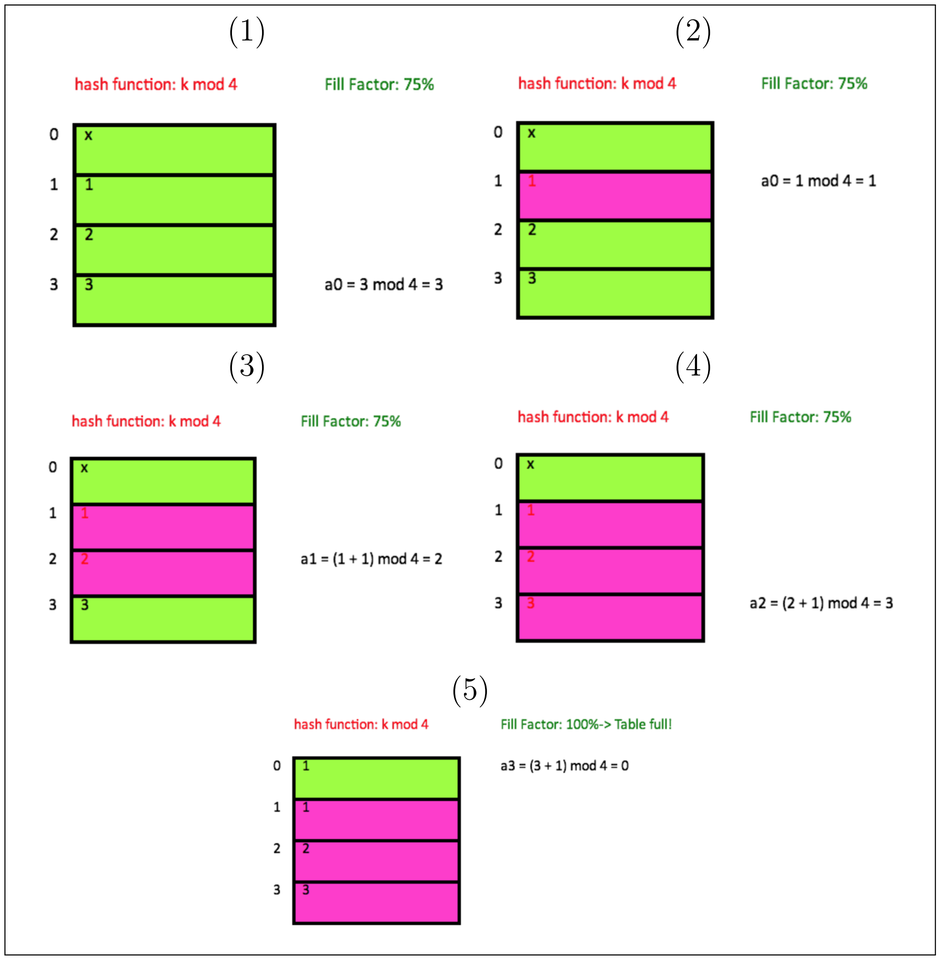


Figure 38: Linear Probing

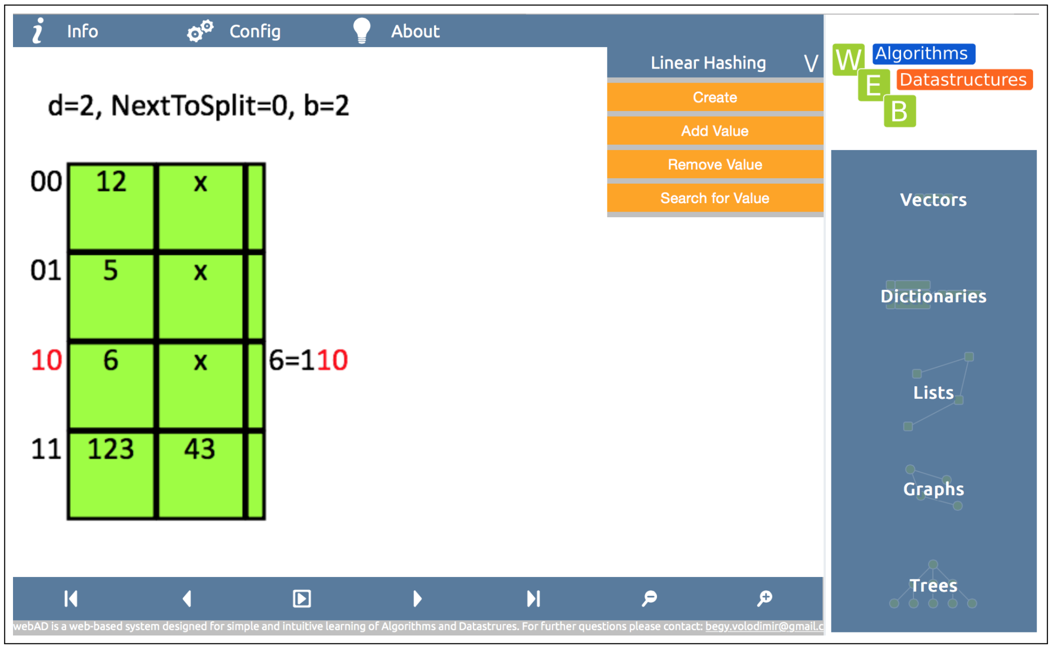
Linear Probing supports same operations as Double Hashing, which function alike in the static/extendible modes:

- Create, producing a table of variable size designated by user in both static and extendible modes.
- Add Value, putting a value if the table is not 100% filled in static mode and without constraints in extendible mode. Analog to Double Hashing, insertion into an extendible table causes it to grow and reorganize all data sets, when more than 70% of table is occupied.

- Remove Value, deleting a value if found and adding the corresponding entry in the collision path.
- Search for Value, seeking for value, using the collision path.

Tape-recorder is operable when the table is not being extended.

Linear Hashing



d=2, NextToSplit=0, b=2

00	12	x
01	5	x
10	6	x
11	123	43

6=110

webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gmail.c

Figure 39: Linear Hashing

Linear Hashing is a dynamic hashing procedure. This means, when a collision occurs, the initial table is extended and some values are rehashed. In this particular type of hashing data is stored in buckets. Furthermore, there is an overflow section, which lines up extra buckets next to the primary table. Buckets of the table are split one after another, the current bucket, which will be split next is also known as NextToSplit (NTS). The indicator d shows how many positions of the binary representation of data are used by the hash function to locate the bucket. Following condition resets the NTS index and increases d:

```
if(NTS==2^d){
  d++;
  NTS=0;
}
```

Listing 36: Relation of NTS and d

Usually the index of the bucket is identified using d positions of the binary representation. However, if the resulting index is smaller than NTS, one extra position is used.
There are different approaches for extending the table, in webAD the most basic one is realized: a new bucket in the primary section is appended each time some bucket (either from primary or overflow area) is overflown.
When some data is hashed, it is shown together with its binary representation, in which the needed for operation positions are marked red.
When a bucket is split, the data within it and its overflow buckets is rehashed, which is animated in webAD:

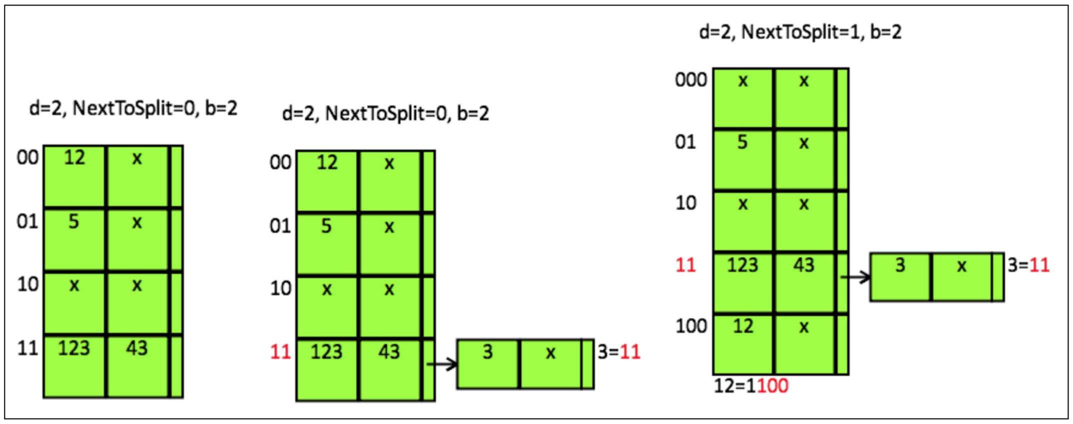


Figure 40: Splitting and Rehashing in Linear Hashing

webAD's visualization of Linear Hashing provides following operations:

- Create, allowing user to specify any Bucket Size b and initial amount of buckets through entering d (amount of buckets is equal to 2^d).
- Add Value, which will add the data into the bucket using either d or $d{+}1$ positions of binary representation.

- Remove Value, which obviously first searches for the desired value. It may also delete an overflow bucket, if it becomes empty.
- Search for Value, using *d* or *d+1* positions in the hash function in same manner as *Add*.

Tape-recorder is always available, except for when a split is being animated.

6.3.3 Graphs

Terminology
A graph is a data structure consisting of two major elements: nodes and edges. There are four types of graphs implemented in webAD:

- Unweighted Undirected Graphs, whose edges cannot be sorted according by weights. When two nodes of such a graph are connected, they are connected bidirectionally.
- Weighted Undirected Graphs, with values assigned to the edges. The default connection between nodes is bidirectional as well.
- Unweighted Directed Graphs, with unprioritized edges, which by default are unidirectional.
- Weighted Directed Graphs, with prioritized edges and unidirectional connections between vertices.

At the bottom of the shrinkable menu with algorithm operations user can always see the adjacency matrix of the current graph.

Breadth First Search
Breadth First Search is a graph traversing algorithm, which shows in which order nodes of a graph are visited given an initial node. For the purpose of determining this order it uses a priority queue, meaning a node, which found itself in the queue first, will be traversed first.
In webAD Breadth First search may be applied on Unweighted Undirected (figure 41) and Unweighted Directed (figure 42) Graphs.
User may choose the art of the graph in *Config* (figure 43).
Nodes, which have already been traversed are colored grey. Vertices in the queue are marked red. The not yet reached nodes are green. In undirected

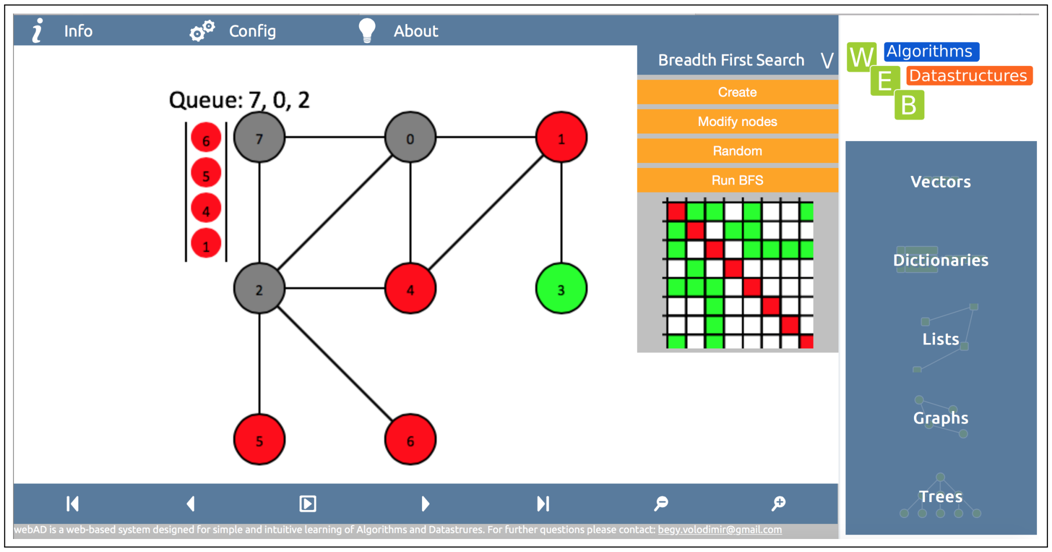

Info
Config
About
Breadth First Search
Create
Modify nodes
Random
Run BFS
Queue: 7, 0, 2
Algorithms
Datastructures
Vectors
Dictionaries
Lists
Graphs
Trees
webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gmail.com

Figure 41: BFS on Unweighted Undirected Graph

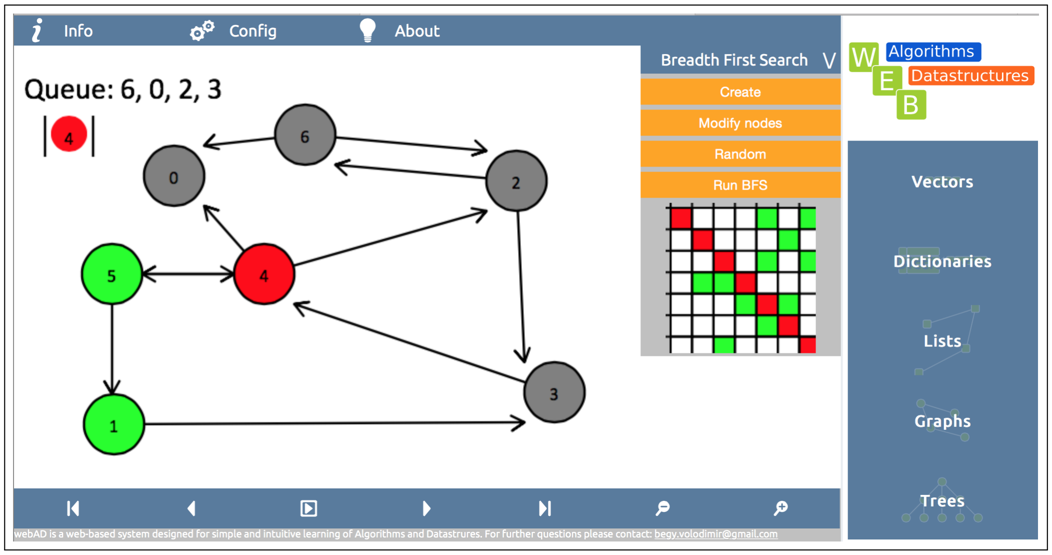
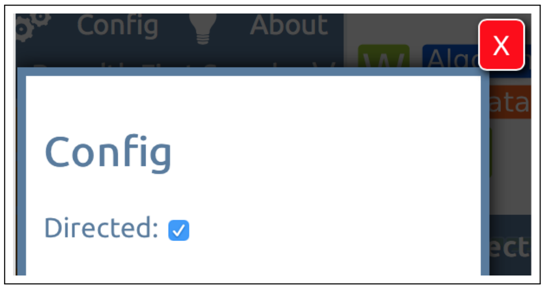

Info
Config
About
Breadth First Search
Create
Modify nodes
Random
Run BFS
Queue: 6, 0, 2, 3
Algorithms
Datastructures
Vectors
Dictionaries
Lists
Graphs
Trees
webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gmail.com

Figure 42: BFS on Unweighted Directed Graph

graphs the node, which is being pointed at by the mouse is appearing orange (in directed graphs a mouse hover has no effect). The initial node is light blue.
This module has following functions in both directed and undirected modes:

- Create, when clicked, the user is asked of how many nodes the graph should consist. Then a popup with adjacency matrix (figure 44) and a textfield for indication of initial vertex appears.
- Modify Nodes, which can be used if the graph is in a state with BFS currently not running (otherwise an error message *"Algorithm already in progress"* surfaces). User has a possibility to add new nodes or change existing edges and initial node.
- Random, creating a truly random graph (not just picking it from hard-coded ones).
- Run BFS, triggering the visualization of the algorithm.

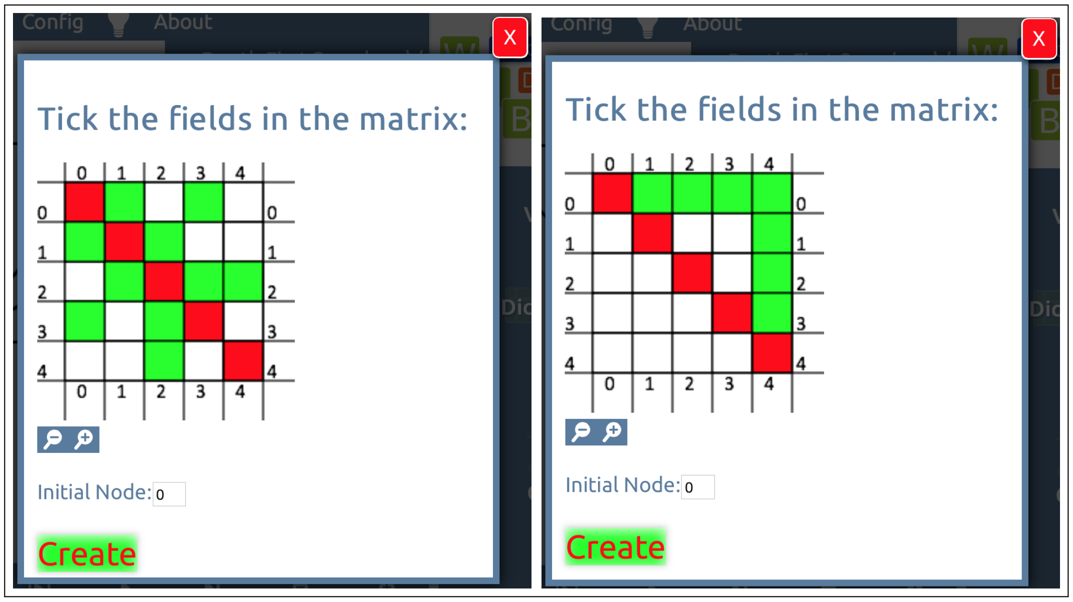


Config

Directed: ✓

Figure 43: BFS Config

The adjacency matrix displays the diagonal in red, user cannot tick these fields and create an edge from a node to itself. If an edge between nodes exists, the field is colored green, otherwise white. In the Undirected Graph, when a certain field from node i to node j is ticked in the matrix, the opposite field from j to i is filled symmetrically. This symmetry is also valid for disconnection of nodes. In directed graphs there is no such symmetry. When user inputs an invalid initial node an error message *"Invalid initial node. It must be between 0 and (matrix size - 1)"* shows up.
When the algorithm is done, it can be re-executed by clicking the *Play* button in tape-recorder.
Tape-recorder is always accessible, but when the type of graph is changed in *Config*, it cannot switch between instances of graphs of different kinds.



Tick the fields in the matrix:

	0	1	2	3	4	
0	🟥	🟩	⬜	🟩	⬜	0
1	🟩	🟥	🟩	⬜	⬜	1
2	⬜	🟩	🟥	🟩	🟩	2
3	🟩	⬜	🟩	🟥	⬜	3
4	⬜	⬜	🟩	⬜	🟥	4
	0	1	2	3	4	

Initial Node: 0

Create

Tick the fields in the matrix:

	0	1	2	3	4	
0	🟥	🟩	🟩	🟩	🟩	0
1	⬜	🟥	⬜	⬜	🟩	1
2	⬜	⬜	🟥	⬜	🟩	2
3	⬜	⬜	⬜	🟥	🟩	3
4	⬜	⬜	⬜	⬜	🟥	4
	0	1	2	3	4	

Initial Node: 0

Create

Figure 44: Adjacency Matrices of Unweighted Undirected and Directed Graphs

Depth First Search

Depth First Search is another graph traversing algorithm. Contrary to Breadth First Search it uses a stack for processing of nodes, which implies, that the vertex, which was added to the stack as the last one, is going to be traversed first.

In webAD it may be applied to Unweighted Undirected and Directed Graphs (which is tunable through *Config*).

The color legend is identical to BFS: initial node is blue, traversed nodes are grey, nodes from the stack are red, not reached vertices are green, the hovered by the mouse in undirected graphs are orange.

Supported operations are as well like those from BFS:

- Create, setting up an unweighted un-directed graph using the adjacency matrix.
- Modify Nodes, allowing to add new nodes, re-arrange existing edges and initial vertex. This feature may be triggered in a state, when DFS is not yet running.
- Random, generating a graph.

- Run DFS, starting the algorithm.

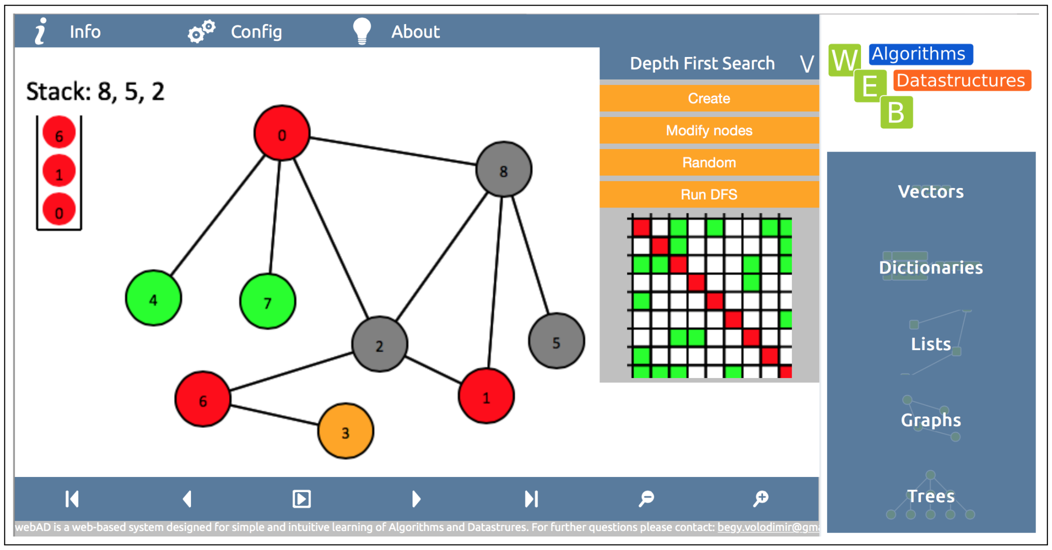
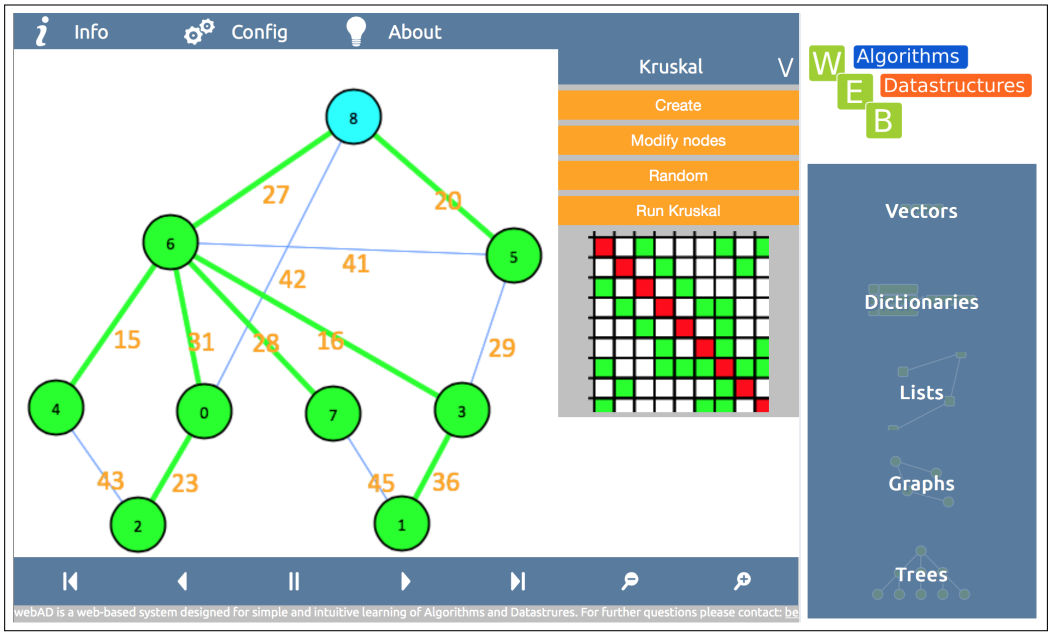

Info
Config
About
Stack: 8, 5, 2
Depth First Search
Create
Modify nodes
Random
Run DFS
Algorithms
Datastructures
Vectors
Dictionaries
Lists
Graphs
Trees
webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gm

Figure 45: DFS on Unweighted Undirected Graph

Analog to BFS, when execution is finished, it can be re-launched by the *Play* button.
Tape-recorder is capable of switching between all states and instances unless the type of graph is changed in *Config*.

Kruskal Algorithm
Kruskal Algorithm is a greedy procedure for detecting a Minimal Spanning Tree within a graph. Due to the greedy nature it selects the next edge with the smallest available weight during each step. Since a tree is an acyclic graph, the paths chosen by the algorithm may not form a circle at any point. In webAD this algorithm is visualized on Weighted Undirected Graphs.
The color legend for the nodes is that of an undirected graph. The weights of the edge are placed in the middle in orange. Additionally the edges are colored while algorithm is running: the not processed edges are black, the ones forming the Minimal Spanning Tree are thick and green, edges causing a cycle and thus not included in the tree are thin and blue.
User may utilize following functions:

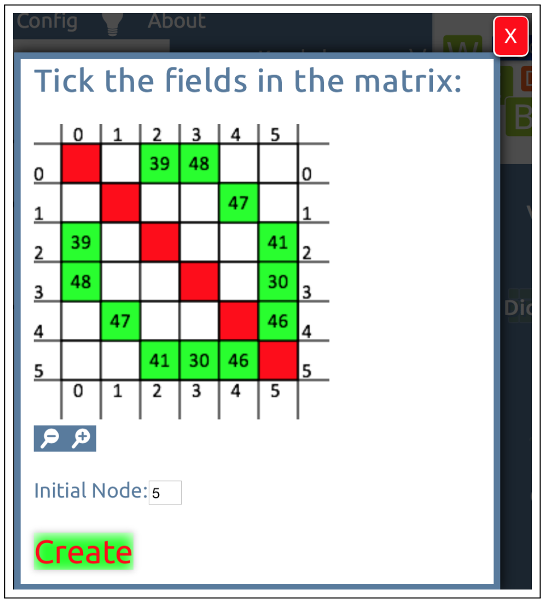
- Create. This function calls the window, which includes the adjacency matrix for the Weighted Undirected Graph and the input field for the initial node.
- Modify Nodes. The feature may be used while a stable state without algorithm running is displayed.
- Random. User may generate a graph with random nodes, edges and weights.
- Run Kruskal. This button launches the visualization.



webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: be

Figure 46: Minimal Spanning Tree detected by Kruskal Algorithm

The adjacency matrix of the Weighted Undirected Graph is having symmetric connections between nodes only. Additionally, when user clicks on a field, the weight of the edge is required. It is limited to 999. Furthermore, the user is asked to enter an initial node. This node is not going to play any role in the algorithm execution, but it will help to check whether at least two nodes are connected by the user. Otherwise an error message *"Connect at least 2 nodes!"* appears.



Tick the fields in the matrix:

	0	1	2	3	4	5	
0	🟥		39	48			0
1		🟥			47		1
2	39		🟥			41	2
3	48			🟥		30	3
4		47			🟥	46	4
5			41	30	46	🟥	5
	0	1	2	3	4	5	

Initial Node: 5

Create

Figure 47: Adjacency Matrix of a Weighted Undirected Graph

The visualization may be re-played when finished by clicking *Play* again. Tape-recorder is usable at any time and switches between any state and instance.

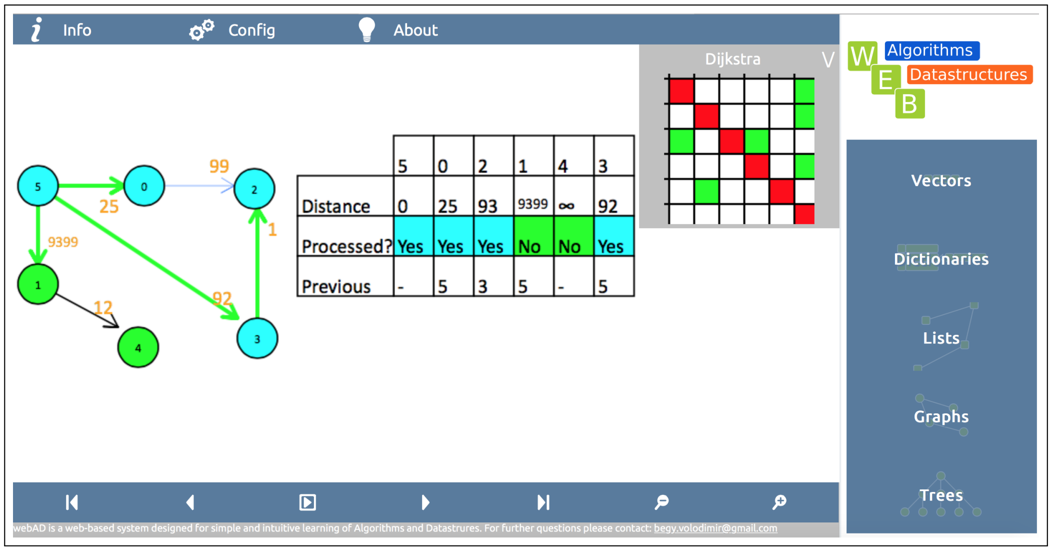
Dijkstra Algorithm

Dijkstra Algorithm is a greedy algorithm for detecting shortest paths to all the nodes within the graph from an initial vertex. Despite being greedy it always delivers optimal results. The only precondition for it to work is that all the edges have positive weights. If all edges have weight of *1*, the order of traversal is identical to Breadth First Search.

In webAD the Dijkstra Algorithm is visualized on Weighted Directed Graphs. The visualization consists of two parts:

- The graph itself. Nodes, which are already processed are colored blue. Otherwise they are marked green. The not yet analyzed edges are black by default and of average thickness. Edges, which are building some shortest path to a certain node are visualized thick and green. Unnecessary edges are represented thin and blue.

- The table with information of traversal. The columns of the table represent all nodes, the first column is always dedicated to the initial node. The rows represent three types of information: distance from initial node to the actual one, whether the current node is fully processed and the previous node forming the shortest path to it (all shortest paths may be identified from the table by following a chain in this row). If a node is not yet reached, the distance to it is marked with an infinity sign (∞). The distance to it is then denoted with a dash (-). If the node is processed, in its corresponding cell it says "Yes" with blue background color. Contrary it says "No" with green background color.



	5	0	2	1	4	3
Distance	0	25	93	9399	∞	92
Processed?	Yes	Yes	Yes	No	No	Yes
Previous	-	5	3	5	-	5

webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gmail.com

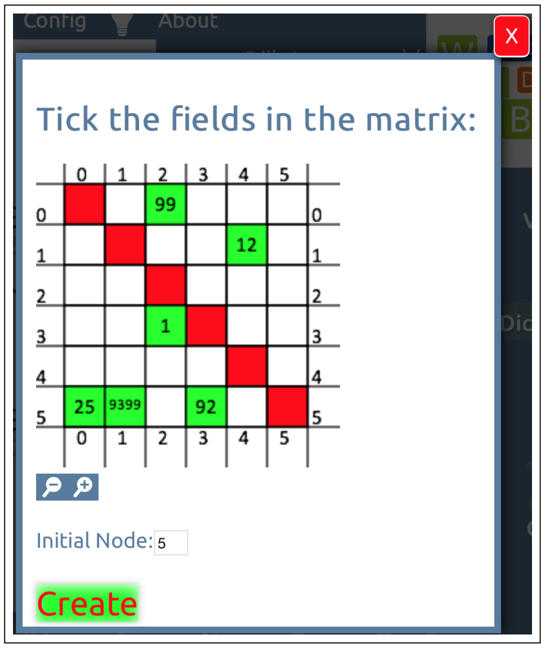
Figure 48: Dijkstra Algorithm

Operations provided by the algorithm, which can be animated are:

- Create. A new graph is created using the adjacency matrix and inputting the initial node. At least two nodes need to be set up and connected through an edge.
- Modify Nodes. First user is asked, whether new nodes are desired. Then this feature calls the same popup, which is used to create a graph. Old relations may be adjusted. To use this option the graph has to be in a state, where Dijkstra Algorithm is not currently operating.

- Random. This function generates a random Weighted Directed Graph with random edges and weights.
- Run Dijkstra. Analog to the *Play* button of the tape-recorder, this button launches the visualization.

The adjacency matrix of the Weighted Directed Graph obviously supports unidirectional relations between nodes. After clicking a certain cell user needs to input a weight between 1 and 9999. Note, how the size of the number is adjusted to the cell's width (figure 49). After activating the *Create* button, it is validated, that at least one more node is connected to the initial one. Otherwise an error message will show up.



Tick the fields in the matrix:

	0	1	2	3	4	5	
0	🟥		99				0
1		🟥			12		1
2			🟥				2
3			1	🟥			3
4					🟥		4
5	25	9399		92		🟥	5
	0	1	2	3	4	5	

Initial Node: 5

Figure 49: Adjacency Matrix of a Weighted Directed Graph

Analog to all other algorithms performed on graphs, the visualization will repeat, when user clicks *Play* or *Run Dijkstra* after animation has finished. Tape-recorder is always operable and can switch between any state and instance.

6.3.4 Trees

Terminology
A tree is a special type of graph, which is acyclic. Just like graphs trees consist of nodes and edges. Edges between 2 unique nodes form *paths*. A node, which initiates the tree is called *root*. A node, which terminates a path is known as *leaf*, while others are *internal nodes*.
Each node has exactly one *parent*, while one parent may have multiple *children*. A root has obviously no parents. Vertices on the same level are called *neighbors*.

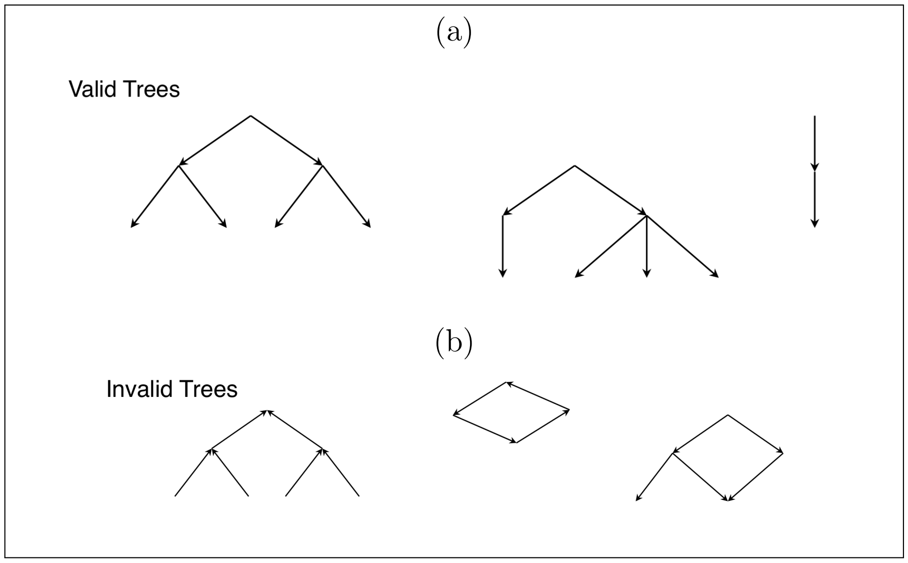
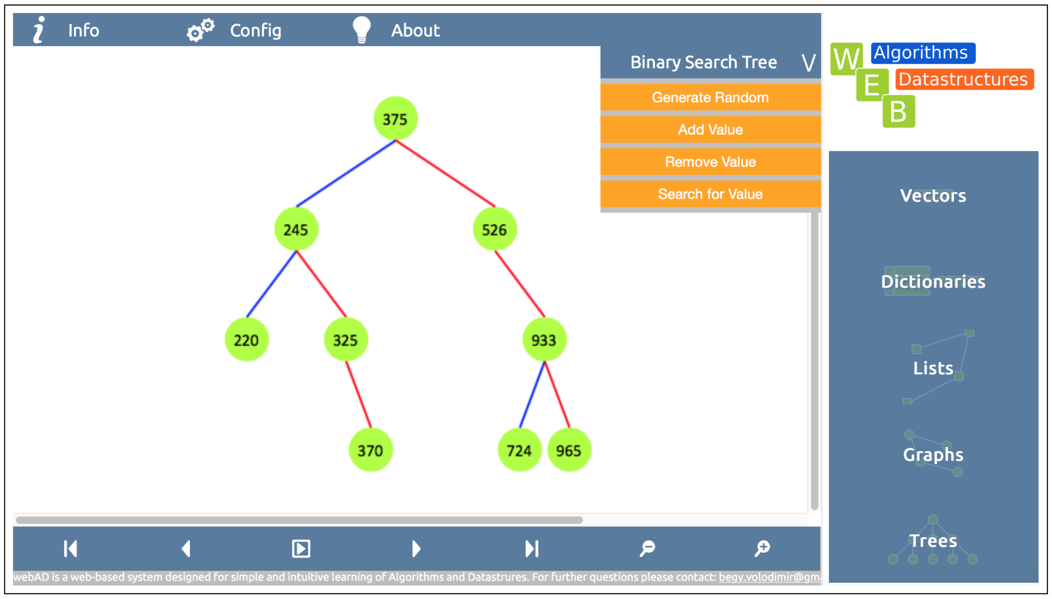


Figure 50: Trees [39]

webAD contains visualizations of some tree data structures (Binary Search Tree, B+ Tree and Binary Min Heap) and a Sorting Algorithm performed on a tree (Heap Sort).

Binary Search Tree
In a Binary Search Tree each node may have at most 2 children. The tree is arranged in such a way, that for any internal node the left child's value is smaller than its parent's, and the right child's content is greater than or equalt to its parent's. It is a dynamic data structure adapted for storing of large data sets.



webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gm

Figure 51: Binary Search Tree

In webAD Binary Search Tree has its color legend. Nodes are generally colored green, but when the animation is showing some path during a certain operation (like insertion or search), the respective node of the path is depicted in coral. In a visualization of a search, the found node is highlighted yellow. The edge from parent to its greater or equal right child is appearing red, to its smaller left child blue. The nodes of the tree never intersect, because an algorithm of the view is calculating how many nodes are needed on each level and expands it accordingly.
User can visualize following operations on the BST:

- Generate Random. This function creates a new tree.
- Add Value. User can insert a number into the tree. This feature is capable of concurrent visualization (figure 52), due to the nature of the Binary Search Tree.
- Remove Value. The operation visualizes all cases of deletion: removal of a node with no children, of a node with only right/left child and of a vertex with two children.
- Search for Value. The search is also animated concurrently (figure 53).

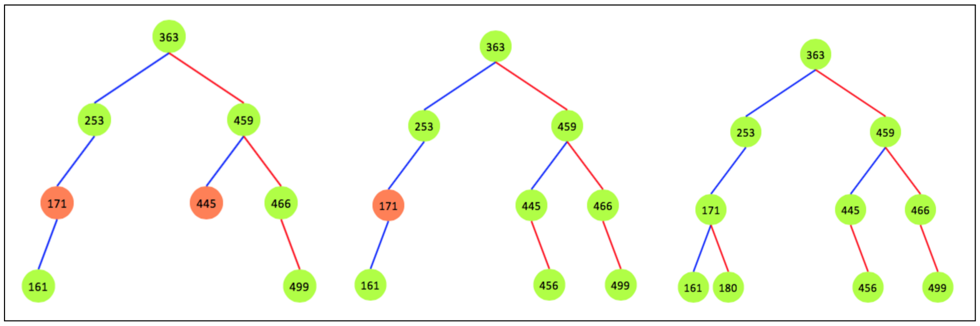


Figure 52: Concurrent Animation of Insertion in BST

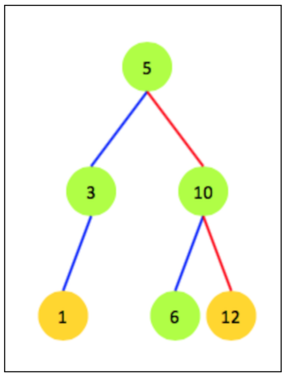
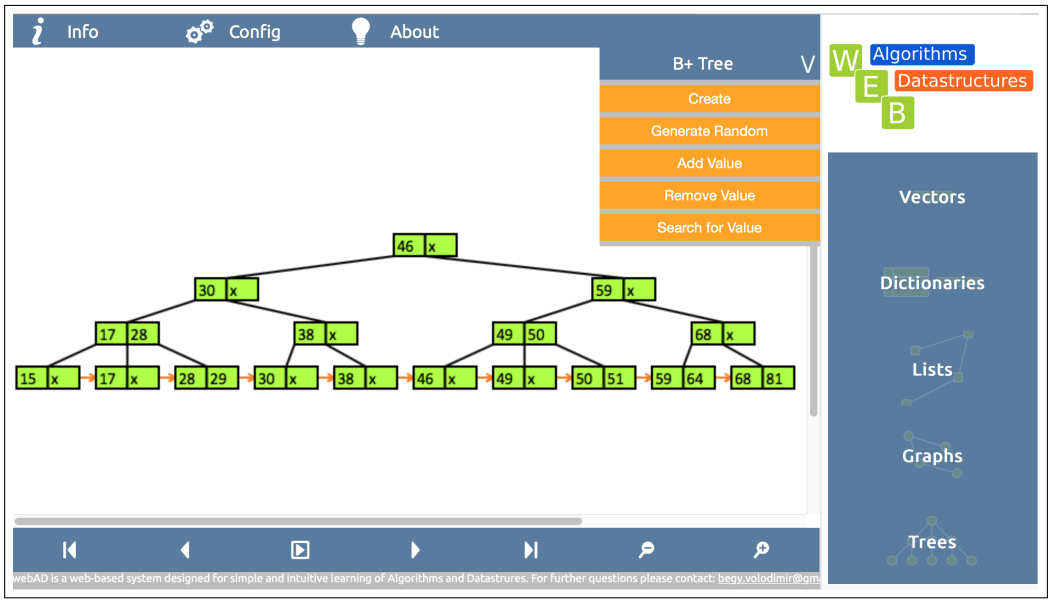


Figure 53: Concurrent Animation of Search in BST

Tape-recorder has no constraints and is always operable, it may interrupt animations.

B+ Tree
B+ Tree is a height-balanced data structure, commonly used by databases. It stores all actual data in the leaves on the lowest level, all other nodes represent indexes, which enable the access on the data.
A B+ Tree has a specific order *k*. Its nodes represent buckets, which are capable of storing multiple data sets or indexes and pointers to children. Within a tree of order *k* all nodes except for the root have to have between *k* and *2k* data sets/indexes. An internal node having *n* keys in it, has to have *n+1* pointers to child nodes. The information on the leaf level is sorted in a specific order (in webAD ascending). When a node overflows on insertion, the tree is extended by splitting. When a node does not meet the constraints

of having between k and $2k$ keys, nodes get merged.
This data structure has proven to be very efficient.



webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gm

Figure 54: B+ Tree of Order k=1

By default nodes are colored green. When some operation is performed and a path is shown, the according nodes and the leading to it edge are marked coral. On the lower level, which stores the actual information in leaves, the pointers between nodes are visualized through orange arrows.
webAD provides following methods for visualization:

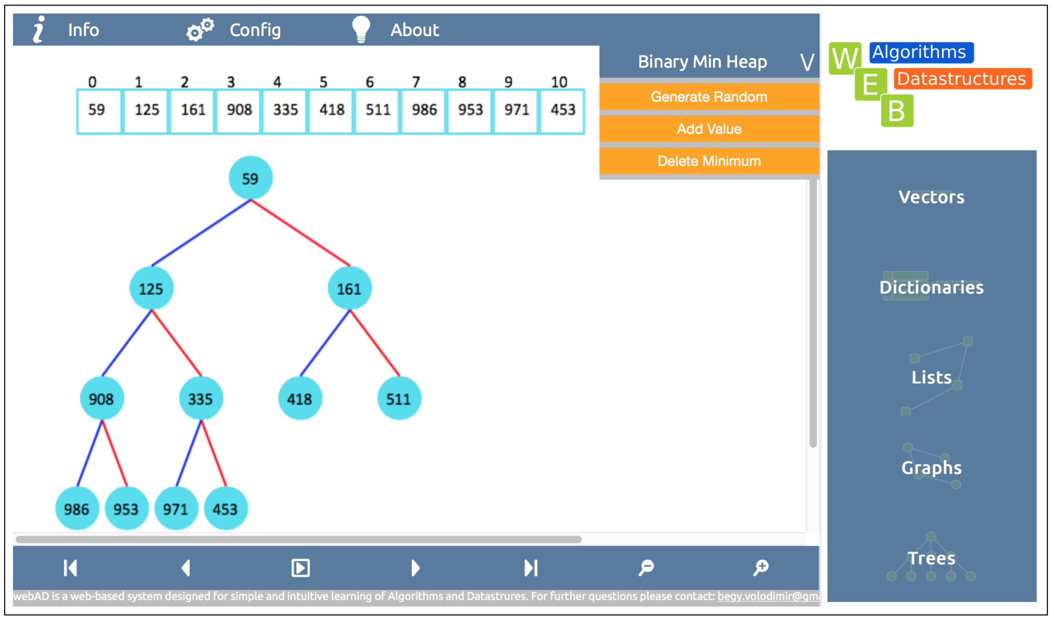
- Create. User may set up a new tree of any order k.
- Generate Random. This feature generates a random tree of random order k.
- Add Value. The input is limited between -*99* and *999* inclusive for view purposes. Contrary to Binary Search Tree the insertion here does not work concurrently, because while some previous value is being added, the path of the upcoming one may easily become completely corrupted because of splits. This is why the execution of any of the functions is interrupted, when any other function is launched. The initial action is aborted so that the tree remains consistent. The insertion visualizes any complex situation, including root splits.

- Remove Value. User has the possibility to remove data and visualize merges of nodes.
- Search for Value. This operation initiates the look up of data in leaves.

Tape-recorder of B+ Tree has no limitations and may switch between any created instance and simulated state.

Binary Min Heap

Binary Min Heap is an unordered tree data structure. The smallest element of the tree is located in the root. On the other hand in the Binary Max Heap the greatest values is in the root. Each node has at most two children, which contain bigger or equal values than the parent (in Max Heap smaller or equal).
A Heap is often implemented using an array. This is also the case in webAD (figure 55).



0	1	2	3	4	5	6	7	8	9	10
59	125	161	908	335	418	511	986	953	971	453

webAD is a web-based system designed for simple and intuitive learning of Algorithms and Datastrures. For further questions please contact: begy.volodimir@gm

Figure 55: Binary Min Heap

The visualization consists of two parts:

- The tree itself. By Default the nodes are blue. When a node is compared with its parent and a swap is needed in order to keep the heap constraints consistent, the two nodes are highlighted purple. If no change is required, they appear green. The edge to the left child is depicted in blue, while the edge to the right child is red.

- The array representation of the tree. The color of the elements of the array are synchronized with the described color legend of the tree.

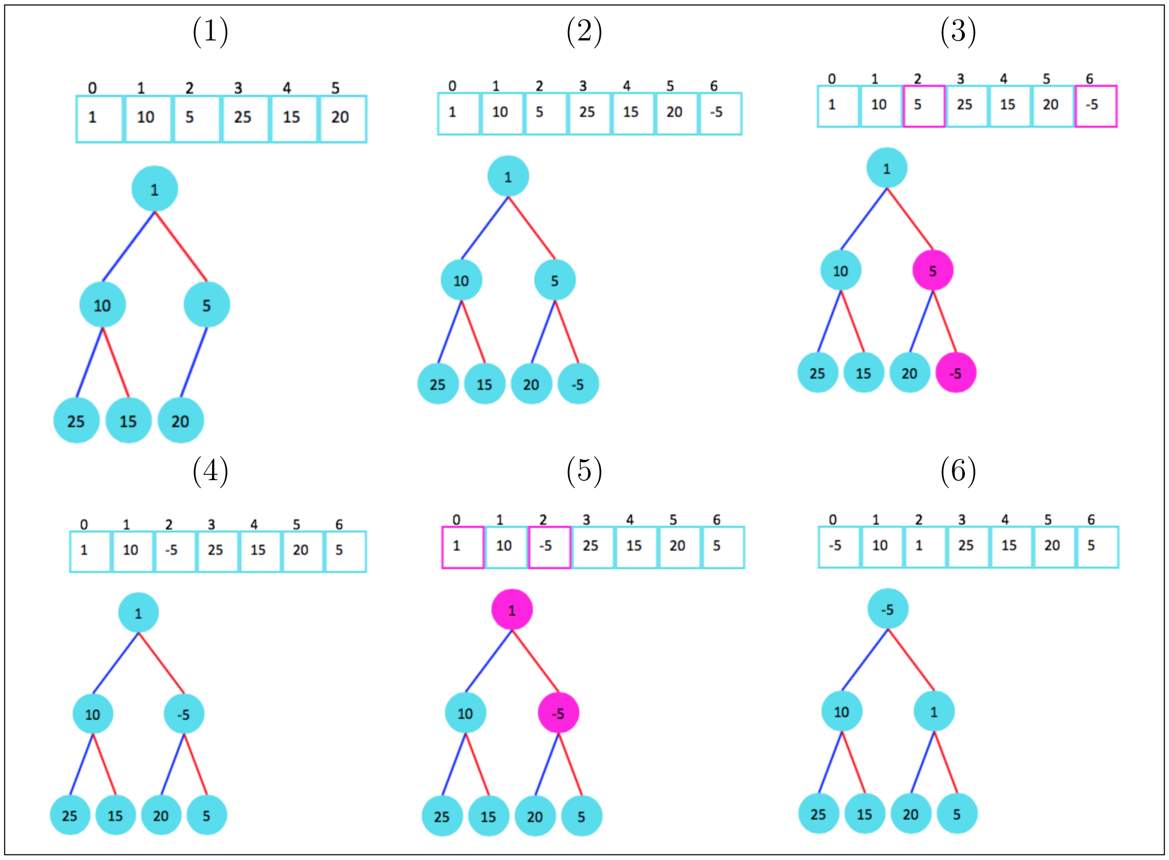


Figure 56: Heap Insertion

User can visualize following operations:

- Generate Random. The feature sets up a new valid Binary Min Heap.
- Add Value. A new node is appended at the end, afterwards the swaps are animated (figure 56).

- Delete Minimum. The root gets removed, and the way the tree gets reconstructed is visualized (figure 57): first the root is swapped with the last element and then deleted. After that the root is compared to its smallest child. If the value of the child is smaller than that of the new root, they get swapped. This operation is repeated until the heap is consistent.

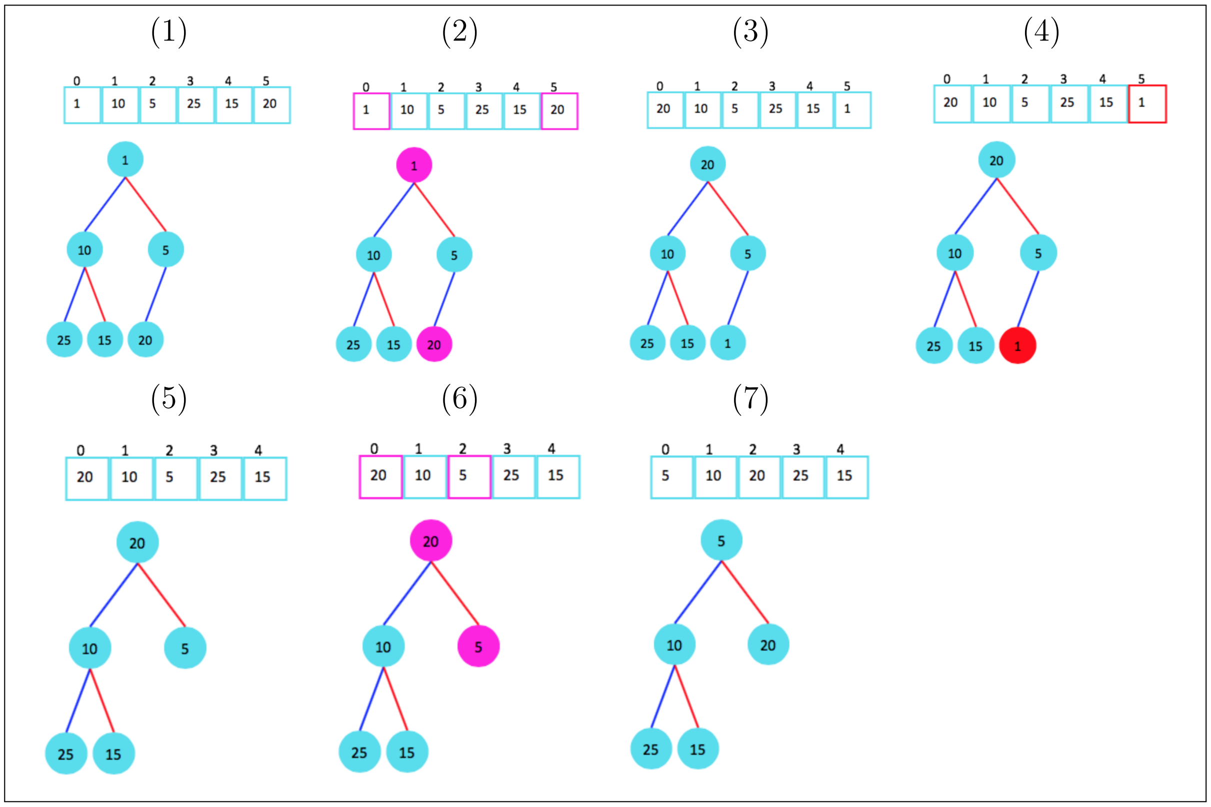


Figure 57: Heap: Delete Minimum

Tape-recorder of the Heap is rather not trivial. It switches between valid heaps only and is available when animation is not in progress.

Heap Sort

A Binary Max Heap can be used for sorting in ascending order (while a Binary Min Heap for the descending order). In webAD the ascending order is implemented.

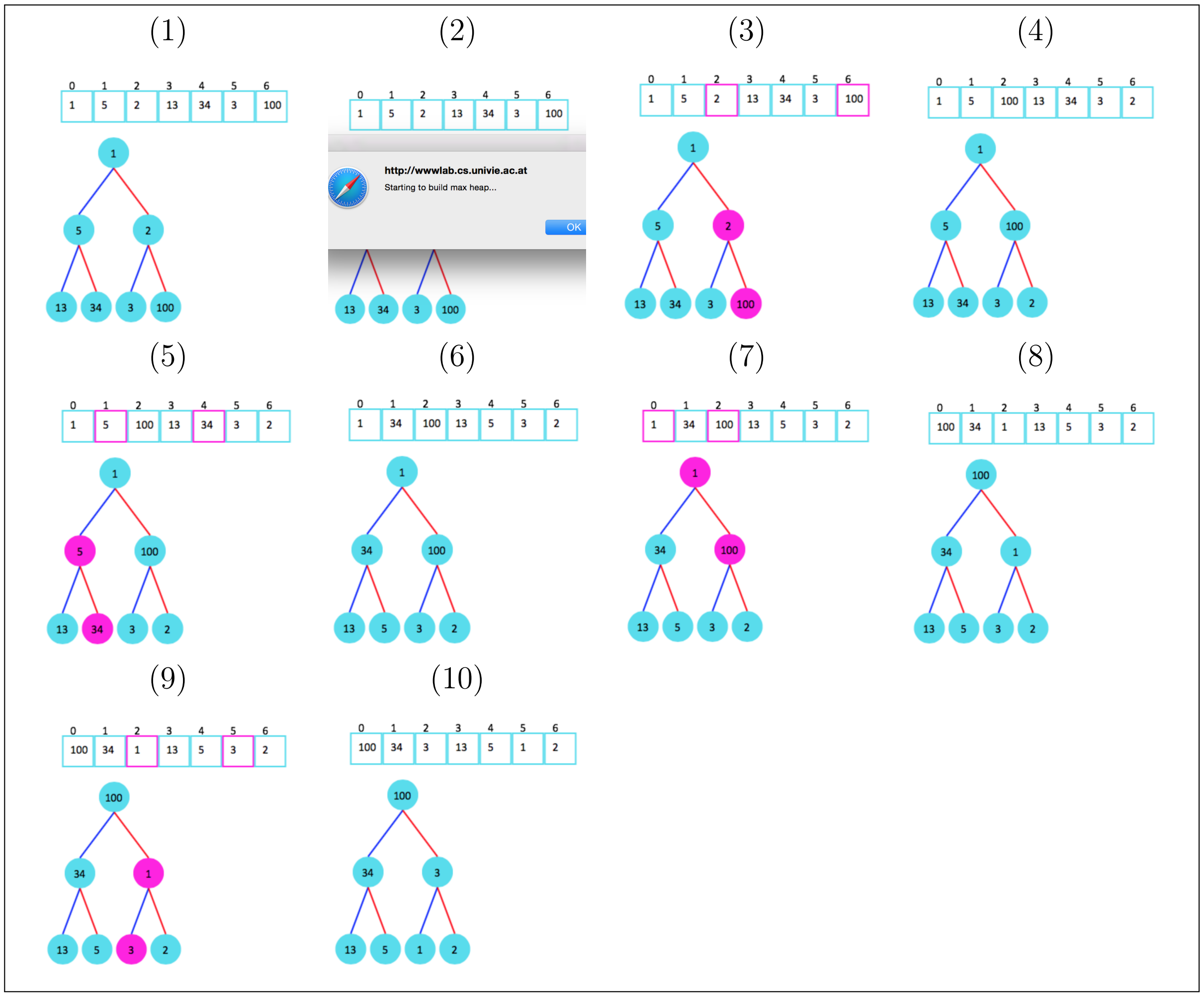
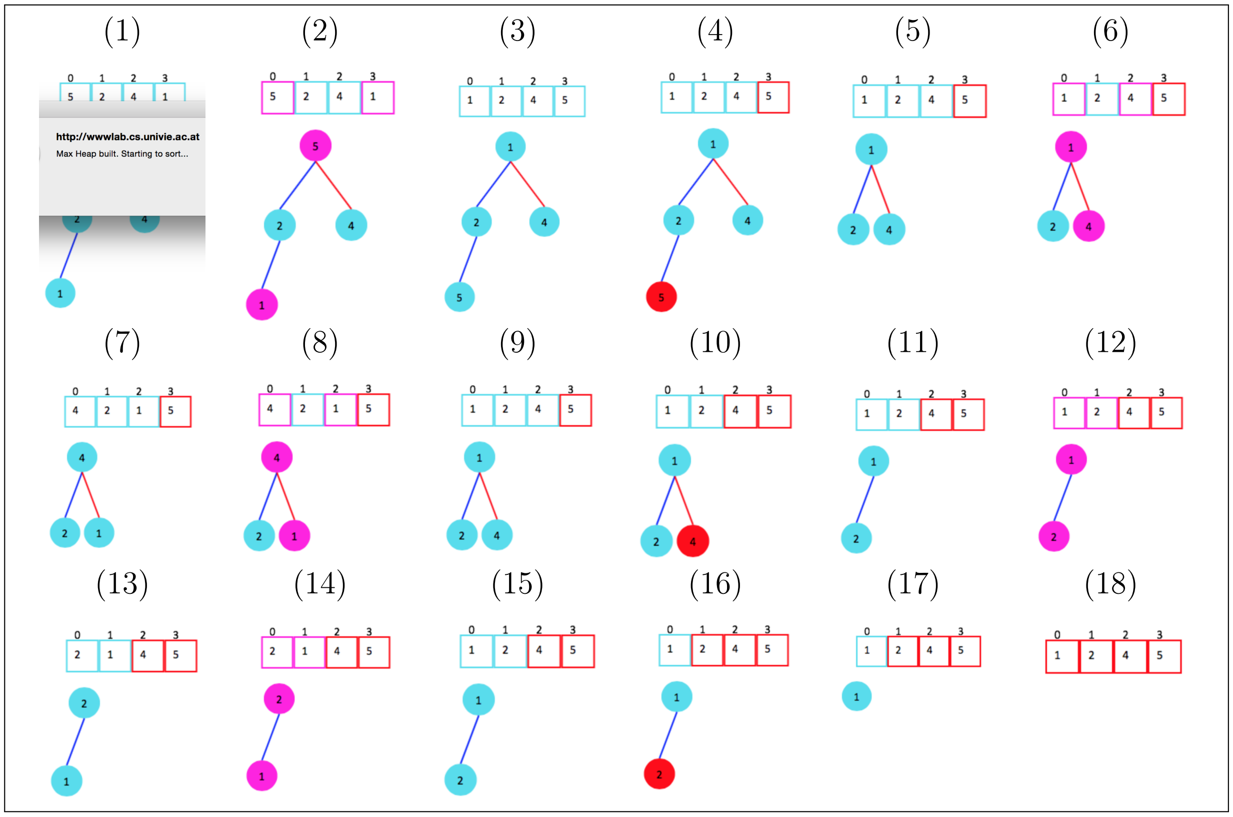


Figure 58: Visualization of BuildMaxHeap()

The sorting consists of two steps:

- *BuildMaxHeap()*. When Heap Sort needs to be applied on some array of numbers, first a Max Heap is built. The procedure takes all parents from the end to beginning and compares them to their biggest child. If the Max Heap property is not satisfied and the child is greater than the parent, the parent sinks. This is repeated recursively until all parents for the initially chosen node are not violating the Heap structure (figure 58).

- Sorting itself. After the Max Heap is built, an iterated procedure begins. During each iteration *i* (starting with *i=0*) the root is swapped with the last element and marked as processed at its new place at index *m-i* (while *m=(size of tree)-1*) in the array and deleted from the tree. After the swap and removal, the new root is checked for the heap property and swapped with its descendants in necessary. Each node of the tree is assigned an iteration (figure 59).


http://wwwlab.cs.univie.ac.at

Max Heap built. Starting to sort...

Figure 59: Heap Sort

webAD allows user to apply following in relation with Heap Sort:

- Generate Random. A user can create a random unsorted array with its representation of inconsistent heap.

- Enter Manually. The array for sorting may be entered in any combination separated by spaces.

Tape-recorder of this module switches between consistent heaps and is operable any time as soon as at least one valid tree is saved. When the user chooses a certain consistent state, the visualization of sorting can be re-played from that point.

7 Findings

After numerous iterations of re-design and implementation of the platform and thorough analysis of other tools plentiful experiences and findings are extracted.
Generally it is hard to make unambiguous suggestions concerning the applied architecture, used technologies and tools, because different approaches imply support of different goals set by different developers. This is why the reflections of this section mostly focus on generic aspects of usability.

7.1 Recommendations

One of the biggest lessons learned by webAD, is that the design matters a lot. The implementation may have great quality from the information system point of view, but repulsive appearance may and will discourage the users. Color is a major part of the appearance. An important law for choosing the palette in accordance to different regions, is that the smaller the area, the harder it is to perceive the color. This is why such spaces should be marked through bright and highly saturated colors, they appear greater. On the other hand, if such colors are applied to large surfaces, they stimulate the cognition in an exaggerated way and should be ignored. For big areas pastel calm colors should be picked [40]. Looking at figure 60, one can see, that this law makes a giant difference.

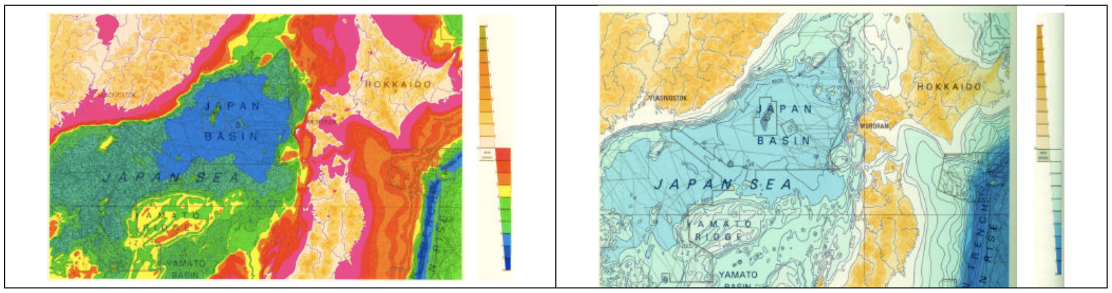


Figure 60: Map with Different Color Palettes [40]

webAD's design was re-arranged following this law, as already shown in figures 2 and 21.
Since we tried to take best care of cognitive offload for the users, it was not the only factor taken into account speaking of colors. Another important

lesson retrieved, is that a limited amount of hues should be used for nominal data, because an average human can differentiate about 8 colors in the view [40]. In webAD this concerns color legends for data structures, for example the choice of the palette for nodes or edges. We recommend to always follow these principles while decorating the user interface and the views.
There are several lessons learned from NetLuke, which have proven to be reasonable in webAD. The amount of the interface components should be minimized. When the platform extends with overflown functionality, it becomes hard to find the needed feature. However, all vital functionality should be visible constantly, users should not strain the nerve in search. In comparison, in NetLuke the operations performed on algorithms/data structures are grouped into the shrinked *Algorithm Control* Menu, while in webAD they are visible by default.
The workflow of the program should be intuitive and lessen the required interaction. For instance in NetLuke after creation or manipulation of the data structure, it should be brought into the algorithm manipulation mode, before an algorithm can be launched. Again, this constantly requires more effort from users. In webAD all such specialities are handled by the system on the code level. Finally, the analog workflows should be consistent in different implemented modules, so that the user does not have to change the habits. These are the three advises we give concerning the integration of workflow.
Concerning the architecture of the platform, an online fat client is recommended for robustness and following of the cloud trend, which are especially valued by mobile devices. JavaScript is a perfect candidate for both purposes. A visualizer should doubtlessly realize the Model-View-Controller architectural pattern to avoid spaghetti code and unclear workflow. This was not the case with webAD, but the mentioned library Backbone.js may be of big interest for its realization in JavaScript. For the drawing of the views KineticJS is obviously a good option, because it provides handy features along with high performance and became a standard. Finally, as mentioned before, jQuery reduces the effort required for manipulation of HTML documents significantly.

7.2 Improvements

Even though webAD has justified our goals and expectations, some design decisions pay off through minor disadvantages. Some of them could be improved using chosen actual technologies:

- The decision to avoid the usage of callbacks for higher performance and browser independence resulted in the fact, that the animations can be paused in consistent states only, because the state of the model is fully reflected in the animation. It would be better for users to be able to pause and resume at any time.

- Several phenomena (for instance split/merging in the b+ tree or complex removal in the binary search tree) could be animated in a more detailed way, showing how they actually occur, which would require vast implementation effort with chosen technologies. Now only the consistent states before and after their occurrence are shown.

- JavaScript has currently no options for persistence. The previously used TaffyDB, promoted as JavaScript Database just provided wrappers for runtime storage objects, which could be accessed in a classic manner of relational database management systems, but could not be persisted. In the current version of webAD no persistence is needed, mainly because we believe, that it violates our minimalistic spirit. If new components, which imply such storage will be desired (for example feedback by users), the introduction of additional technologies will be necessary. A good candidate is PHP, which is developed for browsers and supports database drivers.

- Often operations on models (for example insertion into a certain data structure) have a limited range for values because of view purposes: enormous numbers intersect the drawn borders. Though the allowed intervals are still huge and enable high dynamics, it would be optimal to write view algorithms, which resize the numbers according to their size (a similar algorithm is written for weighted directed graphs) for all modules. The current implementation of system's workflow and models always has limitless dynamics, these ranges are introduced for better look & feel only and can be easily turned off by removing the validation, once the proper view algorithms are implemented.

8 Conclusion and Future Work

webAD has demonstrated, that our design solution fully supports our goals. The visualizer is already used in the *Algorithms and Data Structures* lecture, fulfilling the main task of supporting students and teachers.
The platform is accessible using any desktop or mobile device and any browser and has received some positive acclaims from students.
At this stage, while a major phase of development is completed, there are no unresolved issues with webAD.
It is planned, that further student contributors increase the amount of modules. The existing modules can be extended by more relevant settings in the *Config* section. The chapter *Lists* has no modules at all and should be filled in future. The proposed features from the section *7.2 Improvements* may be considered based on their priority.

References

[1] http://algoviz.org/catalog/hof. Accessed July 18, 2015.

[2] http://jhave.org. Accessed July 18, 2015.

[3] http://aia.cis.unimelb.edu.au. Accessed July 18, 2015.

[4] https://github.com/logos. Accessed September 15, 2015.

[5] http://gruppe.wst.univie.ac.at/workgroups/netluke/. Accessed July 18, 2015.

[6] http://kineticjs.com. Accessed July 18, 2015.

[7] https://jquery.com. Accessed July 18, 2015.

[8] https://www.talater.com/annyang/. Accessed July 18, 2015.

[9] http://backbonejs.org. Accessed July 18, 2015.

[10] http://www.taffydb.com. Accessed July 18, 2015.

[11] http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html. Accessed July 23, 2015.

[12] https://developer.chrome.com/apps/app_frameworks. Accessed July 23, 2015.

[13] https://en.wikipedia.org/wiki/Model-view-controller. Accessed July 23, 2015.

[14] http://b64.io. Accessed July 29, 2015.

[15] http://www.w3schools.com/tags/tag_div.asp. Accessed July 18, 2015.

[16] http://en.wikipedia.org/wiki/Bubble_sort. Accessed July 31, 2015.

[17] http://en.wikipedia.org/wiki/Selection_sort. Accessed July 31, 2015.

[18] http://en.wikipedia.org/wiki/Quicksort. Accessed July 31, 2015.

[19] https://en.wikipedia.org/wiki/Double_hashing. Accessed July 31, 2015.

[20] http://en.wikipedia.org/wiki/Linear_probing. Accessed July 31, 2015.

[21] https://en.wikipedia.org/wiki/Linear_hashing. Accessed July 31, 2015.

[22] https://en.wikipedia.org/wiki/Breadth-first_search. Accessed July 31, 2015.

[23] https://en.wikipedia.org/wiki/Depth-first_search. Accessed July 31, 2015.

[24] https://en.wikipedia.org/wiki/Kruskal's_algorithm. Accessed July 31, 2015.

[25] https://en.wikipedia.org/wiki/Dijkstra's_algorithm. Accessed July 31, 2015.

[26] http://en.wikipedia.org/wiki/Binary_search_tree. Accessed July 31, 2015.

[27] http://en.wikipedia.org/wiki/B%2B_tree. Accessed July 31, 2015.

[28] http://en.wikipedia.org/wiki/Binary_heap. Accessed July 31, 2015.

[29] http://en.wikipedia.org/wiki/Heapsort. Accessed July 31, 2015.

[30] This code snippet was taken from internet, its source is unfortunately lost.

[31] http://stackoverflow.com/questions/3969475/javascript-pause-settimeout. Accessed July 31, 2015.

[32] http://stackoverflow.com/questions/16132905/string-conversion-in-javascript-decimal-to-binary. Accessed July 31, 2015.

[33] `http://www.mkyong.com/java/how-to-determine-a-prime-number-in-java/`. Accessed July 31, 2015.

[34] `http://stackoverflow.com/questions/15628838/kinetic-js-drawing-arrowhead-at-start-and-end-of-a-line-using-mouse`. Accessed July 31, 2015.

[35] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random`. Accessed July 31, 2015.

[36] Begy, V. webAD: Online Visualizer of Algorithms and Datastructures, 2014. Bachelor Thesis.

[37] Clark, R. C., and Mayer, R. E. *E-learning and the science of instruction: Proven guidelines for consumers and designers of multimedia learning*. John Wiley & Sons, 2011.

[38] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[39] Henzinger, M., Schikuta, E., Wanek, H., and Rinderle-Ma, S. VO Algorithmen und Datenstrukturen SS15, 2015. Lecture Slides to Course Algorithmen und Datenstrukturen, University of Vienna.

[40] Möller, T. Encode: Single View Methods. Lecture Slides to Course Databases and Processing of Large Data Sets, University of Vienna. Accessed July 18, 2015.

[41] Prenner, G., Rotheneder, A., and Schikuta, E. Netluke: Web-based teaching of algorithm and data structure concepts harnessing mobile environments. In *Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services* (2014), ACM, pp. 7–16.

[42] Reenskaug, T. Models-views-controllers. *Technical note, Xerox PARC 32*, 55 (1979), 6–2.

[43] Schikuta, E., and Maderbacher, H. Web-based visualization of algorithms and data structures. In *IASTED International Conference Applied Informatics (AI 2001)* (Innsbruck, Austria, 2001), IASTED.

[44] Starner, T. Thick clients for personal wireless devices. *Computer 35*, 1 (2002), 133–135.

[45] Wirth, N. Algorithms and data structures.

List of Figures

Listings

List of Tables

Abstract

Algorithms and Data Structures are the backbone of programming. Thus, they are of vital importance for any computer science student. webAD, standing for web-based visualization of Algorithms and Data Structures is an e-learning platform for students taking the self-titled course *Algorithms and Data Structures* at the University of Vienna. It utilizes the experiences of previous analog projects of the research group Workflow Systems and Technology, such as VADer and NetLuke and extracts the minimalistic mantra: no installation or configuration effort, multi device support, clear structure of didactical content, limitless dynamics of user input and simple extensibility for developers. Compared to other tools for visualization of algorithms and data structures webAD puts a high value on a flexible tape-recorder and implements it with a unique approach. Based purely on HTML5 and JavaScript with the smallest use of external libraries it is designed according to the Model-View-Controller architectural pattern and works as a fat client.

Abstract

Algorithmen und Datenstrukturen sind Grundsätze vom Programmieren. Deswegen sind diese von hoher Bedeutung für alle Informatikstudierende. webAD, bedeutend web-basierte Visualisierung von Algorithmen und Datenstrukturen ist eine E-Learning Plattform für Studierende, die an der Universität Wien das gleichnamige Modul *Algorithmen und Datenstrukturen* besuchen. Sie macht Gebrauch von Erfahrungen der vorherigen analogen Projekten der Forschungsgruppe Workflow Systems and Technology, solchen wie VADer und NetLuke und extrahiert aus diesen die minimalistische Mantra: kein Installations-/Konfigurationsaufwand, Kompatibilität mit verschiedenen Geräten, klare Struktur der didaktischen Inhalte, unbeschränkte Dynamik der Benutzereingabe und einfache Erweitbarkeit für Entwickler. Im Vergleich zu anderen Tools zur Visualisierung von Algorithmen und Datenstrukturen legt webAD viel Wert auf einen flexiblen Tape-Recorder und implementiert diesen mit einem einzigartigen Ansatz. webAD basiert ausschließlich auf HTML5 und JavaScript. Mit der geringsten Nutzung fremder Bibliotheken es ist gestaltet nach dem Architektur-Pattern Model-View-Controller und funktioniert als Fat Client.

Lebenslauf

Name: Volodimir Begy
Geburtsdatum: 19.11.1993

22.06.2011: mit gutem Erfolg im Oberstufenrealgymnasium De La Salle maturiert, Wien

1.10.2013-28.02.2014: Tutor in der Forschungsgruppe Workflow Systems and Technology im Rahmen der Übung Datenbanksysteme, Wien

17.06.2014: akademischer Grad Bachelor of Science (BSc.) in Wirtschaftsinformatik an der Universität Wien